UNIVERSITY INSTITUTE OF COMPUTING

PROJECT Report

on

Password Generator

Program Name: BCA

Linux Administration/23CAP-305

Submitted by:	**Submitted to:**
Name: Anushma Kumari	**Name: - Mr. Rajat Kapoor**
UID: - 23BCA10535	**Designation:**

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

Abstract

In today's digital world, passwords play a vital role in protecting our personal and organizational data. However, the increasing number of cyber threats and hacking attempts has made it essential to generate strong and unpredictable passwords. This project titled **"Password Generator Using Linux Terminal"** focuses on developing a command-line tool that automatically generates complex passwords using random characters, digits, and symbols.

The project aims to demonstrate how security tools can be efficiently built using scripting languages within the Linux environment. The system allows users to specify password length and generates strong passwords that are difficult to crack. It utilizes basic Linux terminal commands or Python's in-built libraries such as random and string to ensure randomness and unpredictability.

This password generator runs entirely offline, ensuring no data leakage to the internet. It's lightweight, fast, and secure, making it ideal for developers, cybersecurity professionals, and system administrators. Overall, this project serves as an introduction to secure automation in Linux environments while promoting best practices in cybersecurity.

Introduction

The need for strong passwords has become more crucial than ever. Weak passwords can lead to data breaches, identity theft, and unauthorized access. A **Password Generator** is a tool that helps users create strong and random passwords that are difficult for hackers to guess. Unlike online generators, this project focuses on building an **offline password generator** using the **Linux Terminal**, ensuring maximum security and user control.

Linux is a popular operating system among programmers and system administrators because of its stability, flexibility, and command-line power. It allows users to write scripts and automate everyday tasks easily. By combining Linux commands or Python scripts, users can generate secure passwords directly in the terminal without any graphical interface.

This project teaches key concepts like randomization, command-line scripting, and encryption principles. It also demonstrates the use of simple yet effective programming constructs to build a useful cybersecurity tool. Through this project, the goal is not only to build a tool but also to understand how security can be automated and integrated into everyday workflows.

Objective of the Project

The primary goal of this project is to **design and implement a Linux Terminal-based Password Generator** capable of generating secure, random, and complex passwords. The detailed objectives include:

1. **To develop a lightweight and efficient password generator:**
   The tool must work smoothly on any Linux distribution without needing heavy software installations or a GUI.
2. **To strengthen understanding of scripting languages:**
   This project helps students gain hands-on experience in Python or Shell Scripting within the Linux terminal environment.
3. **To enhance cybersecurity awareness:**
   The project promotes the importance of strong password creation as a fundamental cybersecurity practice.
4. **To create user customization:**
   Users can define password length and complexity (uppercase, lowercase, digits, symbols).
5. **To automate password generation:**
   Automating this process reduces human error and saves time when creating or managing multiple accounts.
6. **To work offline for privacy and safety:**
   The generator ensures that no data or generated passwords are sent online, maintaining total user privacy.

By fulfilling these objectives, the project proves that even small terminal-based scripts can contribute significantly to secure computing practices.

Tools and Technologies Used

To develop the Password Generator project, several software tools and programming environments were used. Each tool played an essential role in the design, implementation, and testing phase.

1. Operating System: Linux

Linux was chosen because of its open-source nature, stability, and support for command-line scripting. It is ideal for cybersecurity and automation-related projects. Popular distributions like Ubuntu, Kali Linux, and Debian were used for development and testing.

2. Programming Language: Python 3 or Bash Shell

- **Python 3:** Known for its simplicity and readability, Python provides built-in modules like random and string which make password generation easy and secure.
- **Bash Script:** Bash scripting offers a fast and lightweight way to perform the same task using terminal commands and utilities like openssl or /dev/urandom.

3. Text Editors and IDEs

Editors like **VS Code**, **Nano**, and **Vim** were used to write and modify scripts. They offer syntax highlighting, debugging, and terminal integration features.

4. Terminal Emulator

Linux terminal applications like **GNOME Terminal** or **Konsole** were used to execute and test the scripts. They also support command history and automation.

5. Python Libraries Used

- **random:** To generate random characters.
- **string:** Provides a list of predefined character sets.
- **sys/os:** For handling system-level commands and arguments.

Methodology

The Password Generator follows a step-by-step methodology to ensure a secure and structured development process:

1. **Requirement Analysis:**
   The initial phase involved defining user requirements — password length, allowed characters, and ease of use.
2. **Design Phase:**
   A flowchart and pseudo-code were created to define the password generation logic — from taking user input to displaying the final password.
3. **Implementation Phase:**
   Coding was done using Python and Bash scripting. The script was tested for efficiency and randomness of password generation.
4. **Testing Phase:**
   The tool was tested with different lengths and character sets to ensure randomness and no repetition patterns.
5. **Execution Phase:**
   The program was executed on various Linux systems to check compatibility and performance.

Implementation

Python Version

The Python implementation is user-friendly and relies on two main libraries: random and string.

```
import random
import string


def generate_password(length):
    characters = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(random.choice(characters) for _ in range(length))
    return password


length = int(input("Enter desired password length: "))
print("Generated Password: ", generate_password(length))
```

This code allows flexibility in length and ensures a mix of all character types.

Bash Version

For pure Linux scripting enthusiasts, a Bash version is also created:

```
#!/bin/bash
echo "Enter password length: "
read PASS_LENGTH
openssl rand -base64 48 | cut -c1-$PASS_LENGTH
```

This command uses **OpenSSL** to create a base64 random string and then truncates it to the desired length.

Running the Script

1. Save the file (password_gen.py or password_gen.sh).
2. Open Terminal → Navigate to file location.
3. Execute:
   - python3 password_gen.py (for Python)
   - chmod +x password_gen.sh && ./password_gen.sh (for Bash)

Working Process

1. **User Input:**
   The program asks the user for the desired password length (e.g., 10, 12, or 16 characters).
2. **Character Pool Generation:**
   The script builds a pool containing uppercase letters, lowercase letters, digits, and punctuation marks.
3. **Randomization:**
   Using Python's random.choice() or OpenSSL's random generator, a random password is created.
4. **Output Display:**
   The generated password is printed on the terminal securely.
5. **Optional Saving:**
   The password can be optionally copied to the clipboard or stored in a text file.

This step-by-step process ensures complete automation and randomness, providing a new password every time the program is executed.

10. Output (Expanded)

Sample Terminal Output:

```
Enter desired password length: 12
Your Generated Password: qF@8tL!zR2kP
```

Every run gives a unique result such as:

```
Your Generated Password: Vx!3#9aGp7$M
```

Each password includes uppercase, lowercase, symbols, and digits, ensuring strong entropy. Screenshots can be included in the printed report showing different terminal outputs.

Advantages

1. **Security:** Generates random and complex passwords resistant to brute-force attacks.
2. **Offline Access:** Works completely offline — no internet required.
3. **Lightweight:** Uses minimal system resources.
4. **Customizable:** User defines password length.
5. **Platform Independent:** Works on all Linux distributions.
6. **No Data Leak:** Generated passwords remain local to the user.
7. **Simple Usage:** Easy even for beginners.

.Applications

- **System Administrators:** To quickly generate secure passwords for servers.
- **Developers:** To create credentials for applications during deployment.
- **Cybersecurity Students:** To learn about cryptography and randomness.
- **Everyday Users:** For social media, banking, or email passwords.
- **Educational Use:** Demonstrates practical use of Linux and Python scripting.

By applying the same logic, one can create advanced password management systems or integrate it with login portals.

Future Scope

The Password Generator can be expanded with:

1. **Graphical Interface:** Add GUI using Tkinter for user-friendly experience.
2. **Database Integration:** Save generated passwords securely.
3. **Clipboard Copy Feature:** Automatically copy passwords to clipboard.
4. **Password Strength Checker:** Evaluate password complexity.
5. **Cloud Integration:** Store passwords securely in encrypted vaults.
6. **Mobile Version:** Build Android app using Kivy or Flutter.

Thus, the project can evolve into a complete password management application.

Conclusion

This project successfully demonstrates how to generate secure and random passwords directly in the Linux terminal using scripting. It highlights the importance of strong passwords and automation in maintaining cybersecurity. The Password Generator tool is fast, reliable, and efficient, offering a safe offline alternative to online generators.

The implementation of both **Python** and **Bash** versions enhances understanding of scripting languages and their practical uses. The project emphasizes open-source principles and demonstrates that complex security tools can be built with minimal resources.

Through this project, the student gains hands-on experience in Linux command-line interface, scripting, randomization logic, and cybersecurity best practices — essential skills for future developers and IT professionals.