PROJECT REPORT

Project Title: Password Cracking & Credential Attack Suite

Submitted by: Gunta Subham **Department:** Computer Science **Domain:** Cybersecurity / Ethical Hacking

1. Project Overview

This project focuses on designing and developing a practical toolkit for password policy testing and credential security assessment. Weak passwords remain one of the most exploited vulnerabilities in cybersecurity, with attackers frequently using dictionary attacks, credential dumping, and brute-force techniques to compromise systems.

This project provides an ethical, controlled environment to understand how password cracking works, how credentials are stored, and how security teams can reinforce authentication mechanisms. The developed suite includes tools for dictionary generation, hash extraction (Linux/Windows), brute-force simulation, and password strength analysis .

2. Practical Motivation

Passwords remain the first line of defense for user authentication. However, poor password practices lead to severe security risks such as account takeovers, privilege escalation, and data breaches .

This project enables hands-on learning of:

- How password hashes are stored and protected.
- How attackers attempt to crack passwords using various algorithms.
- How defenders can audit and strengthen authentication systems to prevent unauthorized access.

3. Project Objectives

The primary objectives of this project are:

1. **Dictionary Generator:** Develop a tool to create custom wordlists for password testing.
2. **Hash Extraction:** Extract password hashes from Linux shadow files and Windows SAM databases in an ethical environment.
3. **Brute-Force Simulation:** Build a simulation engine to test password robustness against brute-force attacks.

4. **Strength Analysis:** Analyze password strength based on complexity, entropy, and predictability.
5. **Audit Reporting:** Generate detailed audit reports summarizing vulnerabilities and mitigation steps.

4. Practical Scope

The project is divided into several functional modules:

A. Dictionary Generator

- Generates custom wordlists based on user input.
- Includes mutation rules (e.g., uppercase variations, appending numbers, and "leet-speak" substitutions like @ for a) to simulate realistic password patterns.

B. Hash Extraction Module

- **Linux:** Extracts entries from /etc/shadow files.
- **Windows:** Extracts SAM registry hives to identify NTLM hashes.
- **Algorithm Identification:** Identifies hashing algorithms used (e.g., MD5, SHA-256, SHA-512, NTLM).

C. Brute-Force & Attack Simulator

- Simulates dictionary attacks using pre-loaded or custom wordlists.
- Simulates brute-force cracking for short passwords using character permutations (lowercase + digits).
- Provides feedback on "Time to Crack" and success status.

D. Password Strength Analyzer

- Calculates password entropy to estimate resistance against attacks.
- Categorizes passwords as "Weak," "Moderate," or "Strong" based on character sets (lowercase, uppercase, digits, punctuation).

E. Report Generation

- Logs all session activities, including cracking attempts and strength analysis.
- Exports a session audit report (.txt) detailing vulnerabilities found.

5. Tools & Technologies Used

- **Programming Language:** Python.
- **GUI Framework:** Tkinter (for the graphical user interface).

- **Libraries:**
    - hashlib: For hashing operations (SHA-256).
    - itertools: For generating brute-force permutations.
    - threading: To run attacks without freezing the GUI.
    - PIL (Pillow): For handling UI images.

6. Workflow / Architecture

The project follows a linear workflow designed to mimic a real-world security audit:

1. **User Input:** The user provides target credentials, wordlists, or hash files.
2. **Dictionary Generation:** Custom wordlists are built using specific patterns.
3. **Hash Extraction:** The system loads and identifies hashes from system files.
4. **Attack Simulation:** The engine attempts to reverse the hash using Dictionary or Brute-force methods.
5. **Analysis & Reporting:** The results are analyzed for strength and compiled into a final report.

7. Implementation Details

The application is built as a single Python class PasswordAuditApp. Key functions include:

- **identify_hash_algo(hash_str):** A helper function that analyzes hash prefixes (e.g., \$6\$, \$1\$) to identify the algorithm used (SHA-512, MD5, etc.).
- **dictionary_attack():** Iterates through a provided list of words, hashing each one and comparing it to the target hash.
- **brute_force_attack():** Uses itertools.product to generate every possible combination of characters up to a length of 4.
- **password_strength():** mathematically calculates the entropy bits of a password to determine its theoretical difficulty to crack.

8. Learning Outcomes

Developing this project provided deep insights into:

- **Secure Storage:** Understanding why salting and slow hashing algorithms (like Bcrypt) are superior to simple MD5/SHA storage.
- **Ethical Hacking:** Differentiating between "Red Team" (attacking) and "Blue Team" (defending) methodologies.

- **Policy Enforcement:** How to define and enforce strong password policies to mitigate brute-force risks.

9. Conclusion

The "Password Cracking & Credential Attack Suite" successfully demonstrates the vulnerabilities inherent in weak password policies. By simulating real-world attack vectors in a controlled environment, the tool serves as an effective educational platform for understanding authentication security. The addition of reporting and analysis features makes it a functional prototype for basic security auditing.

Here are structured test cases designed to verify every module of your project. I have divided them by functionality so you can systematically check if your code meets the requirements in the PDF.

1. Password Strength Analyzer

Objective: Verify that the entropy calculation logic works and correctly categorizes passwords.

Test Case ID	**Test Scenario**	**Steps to Execute**	**Expected Result**
ST-01	**Weak Password**	1. Go to "Password Strength". 2. Enter password. 3. Click "Analyze".	Output should say **"Weak"** or **"High Risk"** (Entropy < 40).
ST-02	**Strong Password**	1. Enter Tr0ub4dor&3. 2. Click "Analyze".	Output should say **"Strong"** or **"Low Risk"** (Entropy > 60).

Test Case ID	Test Scenario	Steps to Execute	Expected Result
ST-03	**Empty Input**	1. Leave field blank. 2. Click "Analyze".	Application should not crash; ideally, nothing happens or it asks for input.

---

2. Dictionary Generator

Objective: Verify that wordlists are generated and mutations (like adding numbers or symbols) are applied.

Test Case ID	Test Scenario	Steps to Execute	Expected Result
DG-01	**Basic Generation**	1. Go to "Dictionary Generator". 2. Enter admin. 3. Click "Generate & Save".	A .txt file is saved containing admin, ADMIN, admin123, admin!, etc.
DG-02	**Multiple Words**	1. Enter user, test. 2. Click "Generate".	The saved file should contain mutations for **both** "user" and "test".

---

3. Cracking Simulation (Main Module)

Objective: Test if the tool can crack a password using the Dictionary logic and fallback to Brute-Force.

Test Case ID	Test Scenario	Steps to Execute	Expected Result
CS-01	**Dictionary Success**	1. Go to "Cracking Simulation". 2. **Generate** a dictionary for the word soccer. 3. **Load** that dictionary. 4. Enter soccer as the target password. 5. Click "Start".	Console Log: [+] SUCCESS: Password Found: 'soccer'.
CS-02	**Brute-Force Success**	1. Do **not** load a wordlist. 2. Enter a short password: abc (3 chars). 3. Click "Start".	Console Log: Dictionary failed -> Switching to Brute-Force -> [+] SUCCESS: Password Found: 'abc'.
CS-03	**Brute-Force Limit**	1. Enter a long password: abcdefg (7 chars).	Console Log: [-] FAILURE: Password not found. (Because brute force limit is set to 4 chars in code).

Test Case ID	Test Scenario	Steps to Execute	Expected Result
		2. Click "Start".	

---

4. Hash Extraction (Scope B Check)

Objective: Verify the tool correctly reads files and **identifies the algorithm** (MD5 vs SHA-512, etc.).

Since you might not want to use your real system files, create a dummy text file named dummy_shadow.txt with this content:

Plaintext

root:$6$hJ8s9...:18200:0:99999:7:::

olduser:$1$kL5...:18200:0:99999:7:::

Test Case ID	Test Scenario	Steps to Execute	Expected Result
HE-01	**Linux SHA-512**	1. Go to "Linux Shadow Hash". 2. Load dummy_shadow.txt.	Console Log: `User: root
HE-02	**Linux MD5**	1. Same file, look at the second line.	Console Log: `User: olduser
HE-03	**Windows SAM**	1. Create a dummy file dummy_sam.txt: Administrator:500:aad3b...:31d6c...:::. 2. Go to "Windows SAM".	Console Log: `User: Administrator

Test Case ID	Test Scenario	Steps to Execute	Expected Result
		3. Load file.	

---

5. File-Based Dictionary Attack

Objective: Test if the tool can extract a hash from a file and crack it.

Create a dummy file named target_hash.txt with this exact content:

Plaintext

```
CONFIG_FILE_v1

PASSWORD_HASH=5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8

END_CONFIG
```

(Note: That hash is "password" in SHA-256)

Test Case ID	Test Scenario	Steps to Execute	Expected Result
FA-01	**Hash Extraction**	1. Go to "Dictionary Attack (File)". 2. Upload target_hash.txt.	Console Log: Password hash extracted successfully.
FA-02	**Attack Execution**	1. Upload a wordlist containing the word password. 2. Click "Start Dictionary Attack".	Console Log: Password found: password.

---

6. Report Generation (Critical Requirement)

Objective: Ensure the audit log is capturing events and exporting them.

Test Case ID	**Test Scenario**	**Steps to Execute**	**Expected Result**
RG-01	**Generate Report**	1. Perform 2-3 actions from above (e.g., check strength, run a crack). 2. Go to "Generate Audit Report". 3. Verify the preview box has text. 4. Click "Download Report".	A .txt file is saved. Open it and verify it lists the actions you just performed with timestamps.