Secure Login & Authentication System

Terminal-Based Implementation Demonstrating Secure Credential Handling, Multi-Factor Authentication, and Attack Mitigation

Project Planning

Phase 1 - Problem Identification & Initial Design

Identified the need for a login system with secure authentication by having modern controls. **Defined the core components:** password policy enforcement, hashing, MFA, brute-force defence, audit login and simulation of offline password cracking.

Phase 2 - Strengthening Password

- Implemented a Python strength checker based on the length, character variety and entropy.
- Added rejection for commonly used passwords.
- Tested with various inputs; fixed logic issues around scores returning **None**.

Phase 3 - Secure Hashing & Credential Storage

- Integrated **bcrypt** library for hashing and verification, while JSON for persistent storage.
- Fixed decoding errors for byte strings during the hashing process.

Phase 4 - Multi-Factor Authentication (TOTP)

- Added TOTP secret generation and QR code output using **pyotp** and **qrcode** libraries.
- Installed **pillow** library to verify the QR image generation.
- Confirmed the **MFA** workflow using an authenticator app from Playstore.

Phase 5 - Brute-Force Mitigation

Created a lockout mechanism in verification.json that tracks **separate password and OTP failure counters**, along with timestamps to temporarily lock accounts after repeated failures.

Phase 6 - Audit Logging

Built a complete audit logging for **authentication attempts**, **OTP failures**, **account lockouts** and **unknown usernames**. And ensured that timestamps remain readable and consistent.

Phase 7 - Dictionary Attack Simulation

- Implemented offline password cracking using bcrypt.checkpw function.
- Used the **100 most common passwords** from publicly available datasets.
- Performed dictionary attacks on all users and collected cracked results.

Phase 8 - Project Restructuring & Documentation

- Reorganised code into folders:
  components/, **bruteforce/**, **data/**, **auditing/**, **attackSimulation/**, and **qrcodes/**.
- Updated imports and added **sys.path** correction for attackSimulation.
- Wrote README and finalised logbook.

Research Notes

Password Security

- Strong passwords reduce susceptibility to brute-force and dictionary attacks.
- Entropy **($H = L \times \log_2(\text{pool size})$)** is used to measure unpredictability.
- Real-world password statistics show extremely high use of weak patterns.

Secure Hashing

- Bcrypt was chosen due to salt generation, built-in slow hashing, and attack resistance.
- Important for preventing rainbow tables and high-speed cracking.

Multi-Factor Authentication

- **TOTP (RFC 6238**) adds a second factor of resistance to credential theft.
- Secrets and QR codes must be stored securely.

Brute-Force Defence

- Account lockouts are essential for preventing automated online guessing.
- OTP verification attempts are also rate-limited to prevent bypassing password controls during sensitive actions such as password reset.

Attack Simulation

- Offline dictionary attacks illustrate how weak passwords fall instantly.
- Demonstrates the difference between online and offline attack feasibility.

Development Notes

Phase 2 - Passwords Strengthening

To minimise predictable patterns and reduce the success rate of offline cracking

- Enforced **length ≥ 8**, required **uppercase + lowercase + digit + symbol**.
- Entropy threshold was indirectly enforced via character variety.
- Rejected common passwords.

Phase 3 - Bcrypt Hashing

As bcrypt hashing is slow, it makes the cracking impractical at scale. Thus, the following actions were taken:

- Used **bcrypt.hashpw()** with automatic salt generation.
- Stored hashes in JSON for **persistence**.

Phase 4 - TOTP MFA

To mitigate the risk of password compromise, two-factor authentication was implemented.

- The **PyOTP** library was used for generating **time-based codes**.
- The **QRcode** library was used for the **authenticator app** onboarding.

Phase 5 - Bruteforce Lockout System

To reduce the possibility of guessing the passwords drastically by hardening the login surface. Failed password and OTP attempts are tracked independently. Repeated failures in either phase result in a temporary account lock, preventing brute-force attempts across both authentication factors.

Phase 6 - Audit Logging

Every authentication event is recorded to the **audit.log** file for supporting accountability, detection, and analysis.

Phase 7 - Dictionary Attack

In order to provide a practical understanding of attacker capabilities and password weakness an dictionary attack simulation was implemented.

- Used **bcrypt.checkpw()** on a curated list of common passwords.
- Assesses the vulnerability of registered users.

Testing The System

Functional Tests

Correct Password + Correct OTP = Access Granted
Correct Password + Wrong OTP = Denied **After 3 Attempts**
Wrong Password = Denied With Consistent Messaging
Unknown Username = Returns ‘Invalid Credentials’ (**Without enumeration**)

Lockout Tests

Upon entering the **wrong password 5 times**:

- User account **lockout for 30 seconds**
- Further attempts logged but rejected

The system actually verifies the lockout expiry and users’ subsequent ability to log in. Password reset was tested with repeated invalid OTP submissions, and confirmed that the account lockout mechanism is enforced after exceeding the allowed attempts.

Cracking Passwords Testing

- Ran the **dictAttack.py** script against all accounts
- Tested on deliberately weak accounts and verified rapid password cracking
- No match found for strong password accounts
- Confirmed realistic behaviour of offline attacks

Incident Report Testing

Verified the correctness of the following events to be logged as they illustrate forensic value during investigation. Logs include the following:

- Login Failures
- OTP Failures
- Lockout Events
- Successful Logins
- Unknown Usernames

Weaknesses

- JSON is plain text and should be encrypted in real systems.
- MFA secrets are stored locally without encryption, which is acceptable for an educational prototype but would require secure storage in a production system.
- An offline attack shows vulnerability to weak passwords instantly.
- The logging system could be centralised or tamper-resistant.

Ethical Considerations

All testing was performed on a local system. And no real user data or external networks were accessed during the testing phase. The attack simulation, used in this project, was purely for educational analysis.

Reflection

Skills Developed

- Understanding of bcrypt & secure password hashing
- Implemented MFA (TOTP)
- File-based persistent storage & JSON handling
- Brute-force detection logic and time lockouts
- Authentication workflows and user state tracking
- Running offline cracking simulations
- Audit logging & event tracking
- Project structuring & modular security design

Lessons Learned

- Weak passwords remain the highest security risks, and MFA is essential to protect against credential theft.
- Attackers rely heavily on dictionary lists and leaked password data. This is the reason why proper logging is inevitable to detect suspicious behaviour.
- Small implementation mistakes (path errors, encoding issues) can weaken a system.

Career Relevance

This project helped me to understand the secure authentication workflow by practically applying cryptographic hashing, adding multi-factor authentication to avoid unauthorised access. Additionally, defending against brute-force attacks and audit logging has strengthened my skills in secure software design, modular development, ethical security testing and secure account recovery mechanisms. These skills are valuable in my future endeavour of software/security engineering.