FSCT 8561 – Lab 3: Building an OTP Authentication Server

By: Jose Bangate, A01271709

Part 6 – Reflection Questions

1. In order to safeguard user credentials in the event that the system is compromised, hashing is necessary for password storage.
2. By guaranteeing that each authentication code is only valid once and for a very brief period of time, OTP prevents replay attacks.
3. Time-based one-time passwords rely on both parties using the same time window, OTP verification may not work if the client and server clocks are not in sync.
4. OTP-based authentication has a number of drawbacks. Clock drift between the client and server can result in unsuccessful login attempts because it depends on precise synchronization.

Part 7 – Security Analysis

By combining a time-based one-time password (TOTP) with a password factor, the authentication system increases security. Only a per-user OTP secret crated with ***pyotp.random_base32()*** and a SHA-256 password hash ***(pass_hash = hashlib.sha256(...).hexdigest())*** are stored on the server in Auth_server.py. The client in Auth_client.py delivers a SHA-256 hash rather than plaintext and utilizes ***getpass()*** to prevent echoing the password on the screen. This solution satisfies the "never store plaintext passwords" criterion and stops basic credentials leakage from server-side leaks.

OTP mitigated threats: TOTP considerably lessens the effect of a compromised password. An attacker still needs to the current OTP to authenticate even if they figure out the user's password (or password hash). It is more difficult to reuse TOTPs later on because they are subject to frequent changes. Basic replay of older OTPs is lessened by the server's OTP verification ***(totp.verify(otp, valid_window = 1))***, which also allows for slight clock drift while rejecting stale codes.

Remaining attack vectors: The client's transmission of the password hash across the network is the biggest vulnerability. Because the server compares the received hash to the stored hash, an attacker can replay a captured hash as if it were the password. Furthermore, since communication is not encrypted, a local attacker might intercept traffic and instantly obtain the hash secret that is printed to the server console ***([SETUP] otp_secret=...),*** which is sensitive in production but acceptable for a demo. A straightforward lockout after three failures ***(lock_until = now + 30)*** largely mitigates brute force, although targeted phishing or distributed guessing are still feasible.

Why authentication isn't complete security on its own: Even flawless authentication cannot prevent denial-of-service attacks, ensure secure authorization or session management after login, or preserve the confidentiality or integrity of network communications. Secure delivery, secure storage, and controls over the actions of authenticated users are also necessary for a system.

Recommendations for actual deployment: To avoid replay, use TLS for all connections, store passwords using a slow salted hash (such as ***hashlib.pbkdf2_hmac***), and substitute a challenge-response method for "send hash". Stronger rate limitation should be added, OTP secrets should never be printed, and NTP clock sync should be required (as well as suspicious activity should be logged and authenticated).