Day – 4: Password Security & Authentication Analysis

Passwords remain one of the most common ways to authenticate users, but they are also a frequent point of failure in cybersecurity. When stored insecurely or chosen poorly, passwords can be cracked using specialized tools, leading to unauthorized access, data breaches, or worse. Tools like Hashcat and John the Ripper are widely used in ethical contexts to test password strength and demonstrate vulnerabilities.

Hashing vs Encryption:

Password protection relies on hashing, not encryption.

- Encryption is a two-way process: data is scrambled using a key, and the original can be recovered by decrypting with the same (or related) key. It protects data in transit or at rest when you need to retrieve the original content.
- Hashing is a one-way function: it transforms the password into a fixed-length string (the hash) that cannot be reversed to recover the original password. During login, the system hashes the entered password and compares it to the stored hash—if they match, access is granted.

Hashing is preferred for passwords because even if an attacker steals the database, they cannot easily retrieve the originals. Encryption would be dangerous here—if the encryption key is compromised, all passwords are exposed.

Common Hash Types:

Different algorithms produce hashes of varying security levels. Many older systems still use weak ones, making them crackable.

MD5 — 128-bit, very fast, but completely broken (collisions found easily). Insecure for passwords.

SHA-1 — 160-bit, also deprecated due to collision attacks; no longer safe.

SHA-256 / SHA-512 (from the SHA-2 family) — Strong general-purpose hashes, but too fast for password storage without additional protections (they allow rapid guessing).

bcrypt — Designed specifically for passwords; includes built-in salting and a work factor (cost parameter) to make it deliberately slow.

PBKDF2 (Password-Based Key Derivation Function 2) — Iterates a hash function many times (e.g., thousands or millions) to slow down attacks; often uses HMAC with SHA-256 or SHA-512.

Argon2 (especially Argon2id) — The modern winner of the Password Hashing Competition; memory-hard, highly resistant to GPU/ASIC attacks, and the current best practice recommendation.

Modern systems favor Argon2, bcrypt, or scrypt because they are "slow" by design, making mass cracking impractical.

Generating Password Hashes:

You can generate hashes for testing using tools like:

- openssl (e.g., openssl passwd -6 for SHA-512)
- Programming libraries (Python's hashlib, PHP's password_hash())
- Online hash generators (for learning only—never input real passwords!)

Example: Hashing "password123" with MD5 gives 482c811da5d5b4bc6d497ffa98491e38.

Cracking Weak Hashes Using Wordlists:

Attackers use tools like Hashcat (GPU-accelerated, extremely fast) or John the Ripper (versatile, great for many formats and CPU/GPU support) to attempt recovery.

Primary modes include:

- Dictionary attack — Tries entries from wordlists (e.g., RockYou.txt with millions of real leaked passwords).
- Brute force attack — Tries every possible combination (e.g., ?a?a?a?a for 4 characters of any printable ASCII).

Hashcat often outperforms John on GPU-heavy brute-force tasks, while John excels in flexibility and certain hybrid modes.

Brute Force vs Dictionary Attacks:

- Dictionary attack — Uses precompiled lists of common passwords, variations, and leaked credentials. Fast and effective because users often choose predictable words/phrases ("password", "123456", "letmein", or regional terms).
- Brute force attack — Exhaustively tries all combinations (e.g., a-z, 0-9, symbols). Works on short/weak passwords but becomes infeasible quickly as length/complexity increases (e.g., 8 characters with full charset = trillions of possibilities).

Both succeed against weak passwords due to human predictability—people reuse credentials, use common patterns, or pick short/simple ones.

Why Weak Passwords Fail?

Weak passwords fail because:

- They appear in dictionaries or are guessable via patterns (e.g., "Password2025!", name + birth year).
- Fast hashes (MD5/SHA-1) allow billions of guesses per second on modern hardware.
- No/little salting enables precomputed attacks (rainbow tables).
- Reuse across sites means one breach exposes many accounts.

Even strong hashing can't save very weak choices—cracking time drops from years to seconds.

Multi-Factor Authentication (MFA) and Its Importance:

MFA adds a second (or more) factor beyond "something you know" (password):

- Something you have (phone/app token, hardware key)
- Something you are (biometrics)

Even if a password is cracked, MFA blocks access without the second factor. It dramatically reduces risk from credential stuffing, phishing, and cracking.

Recommendations for Strong Authentication:

- Use long passphrases (20+ characters, e.g., "correct horse battery staple").
- Enable MFA everywhere possible (preferably hardware-based like YubiKey).
- Never reuse passwords—use a reputable password manager.
- Choose systems that use Argon2id, bcrypt, or PBKDF2 with high iterations.
- Enforce minimum length/complexity policies, but prioritize length over forced complexity.
- Regularly audit and migrate away from weak hashes (MD5/SHA-1).
- Educate users: Avoid common patterns; treat passwords as critical secrets.

By understanding these concepts and practicing safely (e.g., on your own test systems), you can better appreciate why strong, modern password practices are essential in today's threat landscape.