Password Security & Authentication Analysis

Hashing is a process that converts any input data (password, file, text) into a fixed-length string of characters, called a hash, using a hash function.

It is a **one-way operation**:

- You can create a hash from data.
- You **cannot reverse** a hash back to the original data.

Input: hello
Output: 5d41402abc4b2a76b9719d911017c592

Hashcat is a password recovery and auditing tool used in cybersecurity to test the strength of passwords by attempting to crack password hashes.

It is widely used by:

- Penetration testers
- Ethical hackers
- Security researchers
- Digital forensics professionals

Hash
Password: admin123
Hash (MD5): 0192023a7bbd73250516f069df18b500

What Does Hashcat Do?

Hashcat:

1. Takes a **hash** as input
2. Generates password guesses
3. Hashes each guess
4. Compares the result with the target hash
5. Stops when a match is found

Supported Hash Types

Hashcat supports **300+ hash algorithms**, including:

- MD5 (unsalted hashes/weak)
- SHA-1 / SHA-256 / SHA-512 (unsalted hashes/weak)
- NTLM (Windows passwords)
- bcrypt
- WPA/WPA2 Wi-Fi handshakes
- Linux /etc/shadow hashes

Attack Modes in Hashcat

1. **Dictionary Attack (-a 0)** it is also known as straight attack which is simplify just using a dictionary word list like **rock you.text** .

Hashcat tries passwords exactly as they appear in a wordlist.

How it works

- Reads each word from a file
- Hashes it
- Compares with target hash

Example
Wordlist: rockyou.txt
Passwords tried:
123456
password
admin
welcome

Command
hashcat -m0 -a0 hashes.txt rockyou.txt

Best for

- Real-world leaked passwords
- Beginner labs
- Weak user passwords

Weakness

- Fails if password is not in the list

2. **Combination Attack (-a1)** which is using multiple wordlist Hashcat joins words from **two lists**.

How it works

List1: admin, root
List2: 123, 2024, !
Results:
admin123
admin2024
root!
Command
hashcat -m0 -a1 hashes.txt list1.txt list2.txt

Best for

- Corporate-style passwords
- Username + number patterns

3. **Brute-Force Attack (-a3)** which is simply just trying all different combinations of a character set

Tries ALL possible combinations
This is the slowest but most thorough method.

How it works

- Tries every possible character combination
- Length and charset matter a LOT

Example (4-digit PIN)
0000 → 9999

Command
hashcat -m0 -a3 hashes.txt ?d?l?u?s
Where:
- ?d = digits (0–9)
- ?l = lowercase
- ?u = uppercase
- ?s = symbols

Best for
- Short passwords
- PINs
- Known length

Weakness
- Impossible for long passwords

4. Mask Attack (Smart Brute Force) Pattern-based brute force (VERY important**)**
You tell Hashcat the **structure** of the password.

Example
Password pattern:
Capital + lowercase + lowercase + 2 digits

Mask
?u?l?l?d?d

Command
hashcat -m0 -a3 hashes.txt ?u?l?l?d?d

Best for
- Known password policies
- Huge speed improvement vs brute force

5. Hybrid Attack (Dictionary + Mask) Dictionary + brute force combined
There are two types:
Hybrid 1: Dictionary + Mask (-a 6)
admin + 123

welcome + !

Command
hashcat -m0 -a6 hashes.txt rockyou.txt ?d?d

Used when:

- Users add numbers at the end

Hybrid 2: Mask + Dictionary (-a 7)
2024admin
!password
Command
hashcat -m 0 -a 7 hashes.txt ?d?d rockyou.txt

Hash algorithm (-m) = how the password was hashed
Attack mode (-a) = how Hashcat guesses the password

Weekpass.com download wordlist in firefox (small)

Command **:**
locate wordlists → wordlist actually give you the list of all files that contains commonly used password across the globe and it comes from the kali linux distribution.
locate common.txt → It's a text file containing a list of common words or passwords.
man hashcat → opens the manual page (man page) for Hashcat in Linux.
It is the official, built-in documentation that explains how to use Hashcat, its options, modes, and examples.
echo -n "bob" | md5sum | cut -d' ' -f1 >> hashes.txt → This command converts the word bob into its MD5 hash and saves that hash into a file called hashes.txt.

- echo -n "bob" → sends the word bob (without extra newline)
- md5sum → converts bob into an MD5 hash
- sha1sum → converts bob into an SHA-1hash
- sha256sum → converts bob into an SHA-256 hash
- sha512sum → converts bob into an SHA-512 hash
- cut -d' ' -f1 → keeps only the hash (removes extra text)
- >> hashes.txt → saves the hash into hashes.txt

htpasswd -nbB user hello | cut -d: -f2 >> hashes.txt → bcrypt cannot be generated with a simple Linux command like MD5/SHA.

- Hashcat **mode for bcrypt -m 3200**
- hashcat -m3200 -a0 hashes.txt wordlist.txt

Cat hashes.txt → this command shows what is written inside the file hashes.txt

hashcat -a0 -m0 hashes.txt /usr/share/fern-wifi-cracker/extras/wordlists/common.txt → This command tells Hashcat to try cracking MD5 hashes in hashes.txt using a list of common passwords.

- Hashcat → Starts the Hashcat tool
- -a 0 → Means Dictionary Attack(Hashcat will try passwords exactly as they appear in the list)
- -m 0 → Means MD5 hash type
- hashes.txt → File that contains the hashes you want to crack
- /usr/share/fern-wifi-cracker/extras/wordlists/common.txt →A password list with common passwords
- --show only shows cracked pass
- Each algorithm has a numeric ID (e.g., 0 = MD5, 100 = SHA1, 1400 = SHA256, etc.).
- hashcat -a 0 -m 0 hashes.txt wordlist.txt # MD5
- hashcat -a 0 -m 100 hashes.txt wordlist.txt # SHA1
- hashcat -a 0 -m 1400 hashes.txt wordlist.txt # SHA256
- hashcat -a 0 -m 1700 hashes.txt wordlist.txt # SHA512
- hashcat -m 3200 -a 0 hashes.txt wordlist.txt #bcrypt

What is MFA?

Multi-Factor Authentication (MFA) is a security method that requires two or more independent proofs (factors) to verify a user's identity before granting access.

Instead of relying on only a password, MFA adds extra checks—so even if a password is stolen, attackers are stopped.

The Authentication Factors (Easy to Remember)

MFA combines factors from **different categories**:

1. **Something you know**
    - Password, PIN
2. **Something you have**
    - Phone (OTP app), hardware token, smart card
3. **Something you are**
    - Fingerprint, face scan, iris

How MFA Works

1. User enters **username + password**
2. System asks for a **second factor**
    - OTP from app/SMS

- Push approval
- Biometric scan

3. Access is granted **only if all factors are correct**

Result: Password alone is not enough to break in.

Why MFA Is Important (Core Reasons)

1. Stops Credential Theft

Even if an attacker knows the password:

❌ They still need the second factor

❌ Login fails without the user's device/biometric

2. Protects Against Phishing

Phishing can steal passwords, but:

OTPs expire quickly

Push approvals alert the real user

Hardware keys block fake websites

3. Reduces Impact of Data Breaches

If a database leaks:

Passwords alone are useless

Attackers cannot log in without MFA

4. Essential for Remote & Cloud Access

Used widely in:

Email systems

Cloud dashboards

VPNs

Admin panels

This is why organizations **mandate MFA**.

Recommendations for Strong Authentication

1. Use Multi-Factor Authentication (MFA) Everywhere

- Require **at least two factors** (know + have, or know + are).
- Prefer **authenticator apps** or **hardware security keys**.
- Avoid SMS where possible (SIM-swap risk).

Why: Stolen passwords alone won't grant access.

2. Enforce Long, Unique Passwords

- Minimum **12–16 characters**.
- Require **unique passwords per service** (no reuse).
- Allow passphrases (e.g., *correct-horse-battery-staple*).

Why: Length defeats brute force; uniqueness stops credential stuffing.

3. Use Secure Password Hashing

- Store passwords using **bcrypt**, **Argon2**, or **PBKDF2**.
- Enable **salting** (automatic with modern algorithms).
- Tune **cost factors** to be slow enough.

Why: Slow, salted hashes resist offline cracking.

4. Adopt Password Managers

- Encourage users to use reputable password managers.
- Generate **random passwords** automatically.
- Protect managers with MFA.

Why: Humans can't safely remember many strong passwords.

5. Protect Against Phishing

- Use **phishing-resistant MFA** (hardware keys/WebAuthn).
- Enable **login alerts** and **device checks**.
- Train users to verify URLs and prompts.

Why: Phishing bypasses passwords; strong MFA blocks it.

6. Implement Account Protections

- **Rate-limit** login attempts.
- Use **CAPTCHAs** after failures.
- Temporarily **lock accounts** after repeated failures.

Why: Stops automated guessing and brute-force attacks.

7. Secure Recovery and Reset Flows

- MFA-protect password resets.
- Avoid security questions (easily guessed).
- Use time-limited, single-use reset links.

Why: Weak recovery undermines strong login security.

8. Monitor and Log Authentication Events

- Log successful/failed logins.
- Alert on **anomalies** (new device, location).
- Review logs regularly.

Why: Early detection limits damage.

9. Apply Least Privilege & Step-Up Auth

- Grant only necessary access.
- Require **re-authentication/MFA** for sensitive actions (admin changes, payments).

Why: Reduces blast radius if an account is compromised.

10. Keep Systems Updated

- Patch authentication services and libraries.
- Disable deprecated protocols and weak ciphers.

Why: Prevents exploitation of known vulnerabilities.