Task 4: Password Security & Authentication Analysis

1. How passwords are stored: Hashing vs Encryption:

Hashing:

- A **one-way process**: original password **cannot be reversed**
- Same input → same output
- Used specifically for **password storage**
- Example:
  password123 → 482c811da5d5b4bc6d497ffa98491e38

When users log in, the system:

1. Hashes the entered password
2. Compares it with the stored hash

Encryption:

- **Two-way process**
- Can be **decrypted** using a key
- Used for **data protection**, not passwords

Best practice: Passwords should **never be encrypted**, only hashed.

2. Common password hash types:

When you encounter a hash, its length and character set usually give away its type:

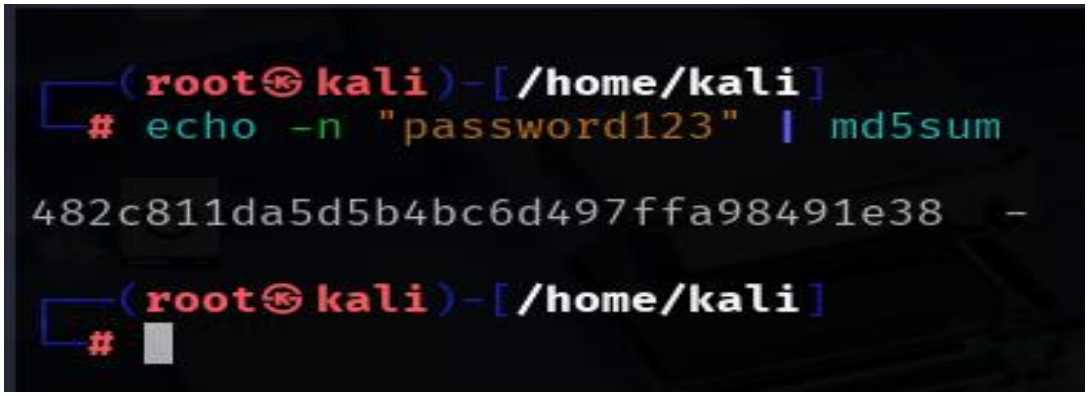
Hash Type	Length (Hex)	Security Level	Common Use Case
MD5	32 chars	**Very Weak**	Legacy systems, file integrity

Hash Type	Length (Hex)	Security Level	Common Use Case
SHA-1	40 chars	**Weak**	Older SSL certificates, Git
SHA-256	64 chars	**Strong**	Modern SSL, Bitcoin, Linux passwords
bcrypt	Varies	**Very Strong**	Modern web apps (includes a "salt" and work factor)

3. Generate Password Hashes (Kali Linux) MD5:

Step 1: Generate MD5 hashes of "password123"

Command:- **echo -n "password123" | md5sum**



Step 2: Create Hash File

- Command:- **nano hash.txt**
- Paste: 482c811da5d5b4bc6d497ffa98491e38 (MD5 hashes)
- Save and exit.

4. Attempt cracking weak hashes:

1. Crack Weak Hashes Using John the Ripper:

Command:- **john --format=raw-md5 hash.txt**

To View Cracked Password:- **john --show hash.txt**

```
┌──(root㉿kali)-[/home/kali]
└─# john --format=raw-md5 hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 128/128 AVX 4×3])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
Proceeding with incremental:ASCII
0g 0:00:03:26  3/3 0g/s 38429Kp/s 38429Kc/s 38429KC/s 1aukmmr..1aukpe5
0g 0:00:10:50  3/3 0g/s 38689Kp/s 38689Kc/s 38689KC/s abhjurt0..abhj11jg
0g 0:00:10:51  3/3 0g/s 38683Kp/s 38683Kc/s 38683KC/s lprakk3m..lpraghep
0g 0:00:10:52  3/3 0g/s 38677Kp/s 38677Kc/s 38677KC/s durnkm3y..durnpet8
```

Cracking hashes using brute force attack taking too much time.

2. Crack Weak Hashes Using Hashcat (used rockyou wordlist):

Command:- **hashcat -m 0 hashes.txt /usr/share/wordlists/rockyou.txt**

```
┌──(root㉿kali)-[/home/kali]
└─# hashcat -m 0 hash.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 18.1.8, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
====================================================================================================================================
* Device #1: cpu-sandybridge-11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz, 1435/2934 MB (512 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 3 secs

482c811da5d5b4bc6d497ffa98491e38:password123

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
```

Hashes are easily cracked in less time by “rockyou.txt” Dictionary Attack.

5. Understand brute force vs dictionary attacks:

Attack Type	Description	Speed
Dictionary	Uses known passwords	Fast
Brute Force	Tries all combinations	Slow

Brute force becomes impractical against strong hashing (bcrypt + salt).

6. Analyze why weak passwords fail:

Password Strength Evaluation (as per "passwordmeter.com")

Password	Composition	Score (Example)	Feedback / Weaknesses
password123	Lowercase + numbers	Weak (43%)	Too common, predictable, dictionary word
Pass@2024	Uppercase + lowercase + number + symbol	Strong (92%)	Some complexity, but still short
S3cur3!Life#2025	Uppercase + lowercase + numbers + symbols + 14 chars	Very Strong (100%)	Long, complex, not in dictionary
qwerty!	Lowercase + symbol, short	Weak (26%)	Common pattern, too short
Tr0ub4dor&3Horse$Battery	Very long passphrase with substitutions	Very Strong (100%)	Hard to crack, highly resistant to attacks

Test Your Password		Minimum Requirements
Password:	password123	• Minimum 8 characters in length • Contains 3/4 of the following items: - Uppercase Letters - Lowercase Letters - Numbers - Symbols
Hide:	☐	
Score:	43%	
Complexity:	Good	

Test Your Password		Minimum Requirements
Password:	Pass@2024	• Minimum 8 characters in length • Contains 3/4 of the following items: - Uppercase Letters - Lowercase Letters - Numbers - Symbols
Hide:	☐	
Score:	92%	
Complexity:	Very Strong	

Test Your Password		Minimum Requirements
Password:	S3cur3!Life#2025	• Minimum 8 characters in length • Contains 3/4 of the following items: - Uppercase Letters - Lowercase Letters - Numbers - Symbols
Hide:	☐	
Score:	100%	
Complexity:	Very Strong	

Test Your Password		Minimum Requirements
Password:	qwerty!	• Minimum 8 characters in length • Contains 3/4 of the following items: - Uppercase Letters - Lowercase Letters - Numbers - Symbols
Hide:	☐	
Score:	26%	
Complexity:	Weak	

Test Your Password		Minimum Requirements
Password:	Tr0ub4dor&3Horse$Battery	• Minimum 8 characters in length • Contains 3/4 of the following items: - Uppercase Letters - Lowercase Letters - Numbers - Symbols
Hide:	☐	
Score:	100%	
Complexity:	Very Strong	

Best Practices Learned

- Use a mix of uppercase, lowercase, numbers, and special characters.

- Ensure length is at least 12–16 characters.
- Avoid dictionary words, common sequences (123, qwerty), or personal info.
- Use passphrases made of multiple random words.
- Enable multi-factor authentication (MFA) for additional security.
- Change passwords periodically and don't reuse across accounts.

Password Complexity Table

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	57 minutes	2 hours	4 hours
6	Instantly	46 minutes	2 days	6 days	2 weeks
7	Instantly	20 hours	4 months	1 year	2 years
8	Instantly	3 weeks	15 years	62 years	164 years
9	2 hours	2 years	791 years	3k years	11k years
10	1 day	40 years	41k years	238k years	803k years
11	1 weeks	1k years	2m years	14m years	56m years
12	3 months	27k years	111m years	917m years	3bn years
13	3 years	705k years	5bn years	56bn years	275bn years
14	28 years	18m years	300bn years	3tn years	19tn years
15	284 years	477m years	15tn years	218tn years	1qd years
16	2k years	12bn years	812tn years	13qd years	94qd years
17	28k years	322bn years	42qd years	840qd years	6qn years
18	284k years	8tn years	2qn years	52qn years	463qn years

7. Study MFA and its importance:

Multi-Factor Authentication (MFA) is a security system that requires more than one method of authentication from independent categories of credentials to verify a user's identity. In the modern digital landscape, it is often cited as the single most effective way to prevent unauthorized access.

1. The Three Core Pillars of MFA

True MFA works by combining at least two of the following "factors." Using two items from the *same* category (like a password and a security question) is technically "two-step verification," not true MFA.

Factor Type	Description	Examples
nowledge	Something you **know**	Passwords, PINs, secret answers.
Possession	Something you **have**	Smartphone (for OTPs), USB security keys (YubiKey), ID badges.
Inherence	Something you **are**	Fingerprints, facial recognition, retina scans.

Advanced Factors: Modern systems also use **Contextual/Behavioural factors**, such as your GPS location, the time of day you usually log in, or even your unique typing rhythm.

2. Why MFA is Crucially Important

- Passwords are the "weakest link" in cybersecurity. MFA acts as a secondary safety net that stops an attack even if the first line of defence fails.
  - **Blocks 99.9% of Attacks:** Microsoft research suggests that MFA can block almost all automated account takeover attempts.
  - **Neutralizes Phishing:** Even if a hacker tricks you into giving them your password via a fake email, they cannot access your account without your physical phone or biometric data.
  - **Combats Password Fatigue:** Since users have dozens of accounts, they often reuse simple passwords. MFA provides a "buffer" that makes even a weak password significantly more secure.
  - **Regulatory Compliance:** Many industries (Finance, Healthcare) now legally require MFA under frameworks like **GDPR**, **HIPAA**, and **PCI-DSS**.
  - **Remote Work Security:** With more people working from home, MFA ensures that employees accessing corporate servers are who they claim to be, regardless of their location.

3. Common MFA Methods (Best to Worst)

Not all MFA is created equal. Some methods are more "phishing-resistant" than others:

1. **Hardware Tokens (Most Secure):** Physical keys like YubiKeys. They require a physical touch and use cryptography that cannot be easily intercepted.
2. **Authenticator Apps:** Apps like Google or Microsoft Authenticator generate time-based codes (TOTP) that don't rely on the cellular network.
3. **Push Notifications:** A "Yes/No" prompt on your phone. Highly convenient, though vulnerable to "MFA Fatigue" (where users click 'Yes' just to stop the notifications).
4. **SMS/Email Codes (Least Secure):** While better than nothing, these are vulnerable to "SIM Swapping" or email account compromise.

8. Recommendations for Strong Authentication:

For Users:

- Minimum **12–16 characters**
- Use password managers
- Unique password per service
- Enable MFA

For Systems:

- Use **bcrypt / Argon2**
- Add **salt**
- Rate-limit login attempts
- Monitor failed logins

9. Final Observation:

- Weak passwords hashed with fast algorithms like MD5 are easily cracked using dictionary attacks in Kali Linux.
- Strong hashing algorithms combined with salting and MFA significantly reduce the risk of credential compromise.
- Password complexity directly impacts resistance against brute force and dictionary attacks.
- Strong passwords are long, unique, and random, not based on predictable patterns.
- Passphrases can be both secure and memorable.
- Combined with MFA, strong passwords provide a robust defense against unauthorized access.