Task 4: Password Security & Authentication Analysis

1. Learn how passwords are stored (hashing vs encryption).

In the world of cybersecurity, **hashing** and **encryption** are the two primary ways to protect data, but they serve completely different purposes.

When it comes to **storing passwords**, hashing is the industry standard. Encryption is almost never used for password storage because it leaves a "backdoor" (the key) that could expose every user if stolen.

1. Hashing (The One-Way Grinder)

Think of hashing like a **meat grinder**. You can put a steak (your password) in and get ground beef (the hash) out, but you can never turn that ground beef back into a steak.

- **How it works:** A hashing algorithm takes an input of any length and turns it into a fixed-size string of characters.
- **Irreversible:** There is no "key" to undo a hash.
- **Deterministic:** The same password will always produce the exact same hash.
- **Verification:** When you log in, the system hashes the password you just typed and compares it to the hash stored in the database. If they match, you're in.

Enhancing Hashing: Salt and Pepper

To make hashes even more secure, developers use two extra "ingredients":

- **Salt:** A random string added to each password *before* hashing. This ensures that even if two users have the same password (e.g., "Password123"), their stored hashes look completely different.
- **Pepper:** Similar to salt, but it is stored in a separate, highly secure location (like a secret vault or different server), making it even harder for hackers to crack the code if they steal the database.

2. Encryption (The Two-Way Vault)

Encryption is like a **locked safe**. You put a message inside and lock it with a key. As long as you have the key, you can unlock it and read the original message again.

- **How it works:** It uses a mathematical key to scramble data into "ciphertext."
- **Reversible:** If you have the decryption key, you can turn the scrambled text back into the original "plaintext."

- **Main Use:** Protecting data **in transit** (like sending an email or credit card info over the web) or sensitive files that someone actually needs to read later.

Key Differences at a Glance

Feature	Hashing	Encryption
Direction	**One-way** (cannot be reversed)	**Two-way** (can be decrypted)
Goal	Integrity & Verification	Confidentiality
Output	Fixed length (e.g., 64 characters)	Variable length (grows with input)
Key Required?	No (uses algorithms)	Yes (requires a secret key)
Best For	**Password storage**, File integrity	Messaging, Emails, Storing files

Why we don't encrypt passwords

If a company **encrypts** your password, they have to store the "key" somewhere. If a hacker breaks into their server and finds that key, they can decrypt **every single password** in the database instantly.

By **hashing** your password, the company doesn't actually know what your password is. Even if a hacker steals the database, they only have a list of "ground beef" hashes that are mathematically impossible to turn back into original passwords.

Modern Password Standards

In 2026, the most recommended algorithms for passwords are **Argon2id** and **bcrypt**. These are "slow" hashes, meaning they are designed to take a tiny bit of time to compute, which makes it incredibly expensive and slow for hackers to try and guess millions of combinations (brute-force).

To see how this works in practice, let's look at **SHA-256**, which is one of the most common hashing algorithms used today.

SHA-256 in Action

Even if you change just one tiny character in a password, the entire hash changes completely. This is called the **Avalanche Effect**.

Input Password	SHA-256 Hash Output
Password123	ef92b778ba7157118197022dbf8c85292376a0ab019245db1d1c546a10056ef9
password123	ef92b778ba7157118197022dbf8c85292376a0ab019245db1d1c546a10056ef9
Password124	62f4e82e666a7b749605658e80554c1555567a57726521db1f4e93c1266205e5

As you can see, changing the 3 to a 4 at the end didn't just change the last digit of the hash—it scrambled the entire string.

The Verification Process

When you create an account, the server doesn't save "Password123." It saves that long string of random-looking characters.

1. **Login:** You type Password123.
2. **Process:** The website runs your input through the SHA-256 "meat grinder."
3. **Comparison:** It checks if the result matches the hash in its database.
4. **Result:** If they match exactly, you are authenticated.

Why this is safer than encryption

If a hacker steals a list of hashes, they can't "decrypt" them to see your password. They would have to try billions of different password combinations, hash each one, and see if any results match yours—a process that can take years for complex passwords.

However, because SHA-256 is actually *too fast* (making it easier for hackers to guess quickly), modern systems now prefer **Argon2** or **bcrypt**, which add "cost factors" to slow the process down on purpose.

2. Identify different hash types (MD5, SHA-1, bcrypt).

To understand the difference between these types, it's helpful to think of them as the **history of security**. As computers got faster, older hashes became "too easy" to crack, leading to the development of more complex systems.

Here is a breakdown of the three most famous types, illustrated with what they look like.

1. MD5 (Message Digest 5)

Created in 1991, MD5 was once the gold standard but is now considered **obsolete** for security.

- **Appearance:** Always 32 characters long, using numbers (0–9) and letters (a–f).
- **The Flaw:** It is **too fast**. A modern computer can try billions of MD5 hashes per second. It is also prone to "collisions," where two different passwords accidentally produce the exact same hash.
- **Current Use:** Mostly used for **checksums** (checking if a file was downloaded correctly without errors), not for passwords.

Example for password "apple123": Hash MD5 is: 49d6051792a6900a658957827878696c

2. SHA-1 (Secure Hash Algorithm 1)

Designed by the NSA and released in 1995, SHA-1 was intended to be the successor to MD5.

- **Appearance:** Always 40 characters long.
- **The Flaw:** Like MD5, it is now considered **broken**. Major tech companies (Google, Microsoft) stopped trusting SHA-1 years ago because it is vulnerable to high-powered "brute-force" attacks.

- **Current Use:** Still found in legacy software and some version control systems (like Git), but being phased out in favor of SHA-256 or SHA-3.

Eg. for password "apple123": SHA-1output: 42701b9701764673322d7c49924403759972b22a

3. bcrypt

This is the **modern standard** for password storage. It was designed specifically to be "expensive" for a computer to calculate.

- **Appearance:** Usually 60 characters long. It looks different because it includes "metadata" directly in the string.
- **The Strength:** It has a **Work Factor** (cost). You can tell the algorithm to run 10,000 or 100,000 loops to generate the hash. This makes it incredibly slow for hackers to guess, but still fast enough for a single user to log in.
- **Built-in Salt:** Unlike MD5 or SHA-1, bcrypt **requires** a salt and stores it inside the hash itself.

Anatomy of a bcrypt hash:

If you look at a bcrypt hash, it tells you exactly how it was made:

$2a$12$R9h/cIPz0gi.URNNX3kh2OPST9/PgBkqquzi.Ss7KIUgO2t0jWMUW

- **$2a$**: The version of bcrypt used.
- **12**: The **Cost Factor** (it ran 2^{12} iterations).
- **R9h/cIPz0gi.URNNX3kh2O**: This is the **Salt** (random data).
- **PST9/PgBkqquzi.Ss7KIUgO2t...**: This is the actual **Hashed Password**.

Comparison Summary

Hash Type	Length	Security Level	Best Use Today
MD5	32 chars	❌ **Broken**	File integrity (Non-security)
SHA-1	40 chars	⚠️ **Weak**	Legacy systems / Git IDs
bcrypt	60 chars	✅ **Excellent**	**User Passwords**

Identifying an unknown hash is a common task in cybersecurity. Since different algorithms produce strings with distinct lengths and character sets, you can often "fingerprint" them just by looking at their structure.

Here is how you can identify the three types we discussed, plus the professional tools used to do it automatically.

1. Manual Identification (The Visual Test)

If you are looking at a string of text, check these three "tells":

- **Is it 32 characters of Hex (0-9, a-f)?** It's likely **MD5**.
- **Is it 40 characters of Hex?** It's likely **SHA-1**.
- **Does it start with $2a$, $2b$, or $2y$?** It is definitely **bcrypt**.

The "Cost" Factor in bcrypt

Because bcrypt is designed for security, it actually encodes its "settings" right into the string. This is helpful for developers because if they want to increase security later, the system can see which passwords were made with the "old" (faster) settings and update them.

2. Using Automated Tools

In a professional setting (like a "Capture the Flag" competition or security audit), you wouldn't guess. You would use a specialized tool.

Hash-Identifier (CLI Tool)

Most security professionals use a tool called hash-identifier (standard in Kali Linux). You simply paste the string, and it gives you the most likely algorithm.

Example Input: 49d6051792a6900a658957827878696c **Output:** ```text Possible Hashs: [+] MD5 [+] Domain Cached Credentials - MD4(NT)

3. Generate password hashes.

In Linux, generating password hashes is a common task for sysadmins setting up user accounts in scripts or for manual /etc/shadow entries. By 2026, the industry has shifted away from older algorithms like MD5 in favor of **SHA-512** and **Yescrypt**.

Here are the three most professional ways to generate these hashes from the command line.

1. Using *mkpasswd* (The Standard Utility)

The *mkpasswd* tool is part of the whois package. It is the most direct way to create hashes that match the Linux /etc/shadow format.

Installation

If the command is missing, install the package:

- **Ubuntu/Debian:** sudo apt install whois
- **Arch:** sudo pacman -S whois

Procedure & Commands

To generate a **SHA-512** hash (Industry Standard):

Bash $ mkpasswd -m sha-512

- **Procedure:** After typing the command, it will securely prompt you for a password. It automatically generates a random **salt** and displays the final hash.
- **Example Output:** 6randomsalt$hashedstring... (The 6 indicates SHA-512).

To generate a **Yescrypt** hash (Modern Default):

Bash $ mkpasswd -m yescrypt

```
┌──(kali㉿kali)-[~]
└─$ mkpasswd -m MD5
Password:
$1$YMmoryMG$FbqNa.nhJkkVutm6xSM94/

┌──(kali㉿kali)-[~]
└─$ mkpasswd -m sha-512
Password:
$6$x2Y97V/ERwwRUyLS$tcQVFldkBxz1bl8aZ6q3Q/pyqgZ7gHuVu7a1FFnWx.nZZsWZsxuVRaDzJ2F7.z.2RGmQaexu.HmdWCzYUrGQ//

┌──(kali㉿kali)-[~]
└─$ mkpasswd -m yescrypt
Password:
$y$j9T$.VWmWKv3z/dTtRtam976c.$USOZ6WXdw3N6VChldwPlX.yOcFNenspFgNvA6e6iT54

┌──(kali㉿kali)-[~]
└─$
```

2. Using openssl (The Universal Way)

Since openssl is installed on almost every Linux system by default, it is a great "no-install" alternative.

Procedure & Commands

To generate a **SHA-512** salted password:

Bash $ openssl passwd -6

- **Procedure:** It will prompt you to enter and verify the password.
- Manual Salt: If you want to specify your own salt, use:

openssl passwd -6 -salt mysalt123 MyPassword

```
┌──(kali㉿kali)-[~]
└─$ openssl passwd -6
Password:
Verifying - Password:
$6$yoQ44aol/Lr3neaA$RVCY8azycpSQGd9om9wd6IfCjC6zUbIsG5Bfv98aSO86Ne1NihCgsBzQx4A.lI6Gh0s629m4A4OtRQa7E/j.g.

┌──(kali㉿kali)-[~]
└─$
```

3. Using python3 (The Developer Way)

If you don't have the utilities above, you can use Python's crypt or bcrypt libraries, which are standard in most distributions.

Procedure & Commands

Run this one-liner to generate a **SHA-512** hash:

Bash $ python3 -c 'import crypt,getpass; print(crypt.crypt(getpass.getpass(), crypt.mksalt(crypt.METHOD_SHA512)))'

Identification Summary

When you see the output, the first few characters tell you which algorithm was used:

Prefix	Algorithm	Security Level
1	MD5	❌ **Obsolete**
5	SHA-256	⚠️ **Older Standard**
6	**SHA-512**	✅ **Very Strong**
y	**Yescrypt**	✅ **Modern Default**
$2b$	**bcrypt**	✅ **Highly Secure**

4. Attempt cracking weak hashes using wordlists.

John the Ripper (JtR) is one of the most popular password security auditing and recovery tools. In a Linux environment, it is commonly used by system administrators to identify users with weak passwords that could be vulnerable to brute-force or dictionary attacks.

Here is a detailed procedure for using John the Ripper for educational and defensive purposes.

1. Understanding the Target Files

In Linux, user account information and passwords are not stored in a single file for security reasons.

- **/etc/passwd:** Contains user account information (UID, GID, home directory) but the password field is usually marked with an x.
- **/etc/shadow:** Stores the actual hashed passwords and is only readable by the root user.

To crack these passwords, JtR needs a combined format of these two files, which we create using a process called "unshadowing."

2. The Cracking Procedure

Step 1: Combine Passwd and Shadow Files

First, you must merge the files into a format JtR understands. You will need root or sudo privileges to read the shadow file.

Bash *$ sudo unshadow /etc/passwd /etc/shadow > myhashes.txt*

This creates a file named myhashes.txt containing the usernames and their corresponding password hashes.

Step 2: Choose a Cracking Mode

John the Ripper offers several modes depending on how much time you have and the complexity of the passwords.

A. Single User Mode (Fastest)

This is the best place to start. It uses the information in the username (like "jsmith") to guess the password (like "jsmith123" or "smithj").

Bash *$ john --single myhashes.txt*

B. Wordlist Mode (Dictionary Attack)

This compares the hashes against a list of common passwords. You can use the default wordlist provided with JtR or your own (like the famous rockyou.txt).

Bash *$ john --wordlist=/usr/share/john/password.lst myhashes.txt*

C. Incremental Mode (Brute Force)

This mode tries every possible combination of characters. It is guaranteed to find the password eventually but can take years for complex strings.

Bash *$ john --incremental myhashes.txt*

Step 3: View the Results

While John is running, you can press any key to see the current status. Once the process is finished (or if you want to see what has been found so far), use the --show flag:

Bash *$ john --show myhashes.txt*

3. Security Best Practices

If you are performing this audit on your own system and find that passwords are easily cracked, consider these mitigation steps:

- **Enforce Password Complexity:** Use the pam_pwquality module to require a mix of uppercase, lowercase, numbers, and symbols.
- **Use Strong Hashing:** Ensure your system is using a modern hashing algorithm like **SHA-512** or **Yescrypt** rather than older, weaker methods like MD5 or DES.
- **Implement MFA:** Multi-Factor Authentication (MFA) provides a second layer of security even if a password is compromised.

Important Note: Always ensure you have explicit, written permission before performing security testing on any system that you do not personally own.

5. Understand brute force vs dictionary attacks.

In cybersecurity, **Brute Force** and **Dictionary Attacks** are the two most common ways hackers try to "guess" our password. While they both have the same goal—matching a plain-text password to a stolen hash—the logic they use is very different.

1. Brute Force Attack

A Brute Force attack is a **mathematical trial-and-error** method. The computer attempts every possible combination of characters until it finds the right one.

- **Logic:** Exhaustive search. It doesn't "guess" based on logic; it simply tries every permutation.
- **The Procedure:**
    1. The attacker identifies the hash type (e.g., MD5).
    2. The software starts at "a", then "b", then "c"...
    3. It moves to "aa", "ab", "ac"...
    4. It continues until it hits "Password123!".
- **Example:** If your password is Dog1, a brute force tool will try:
    - aaaa
    - aaab ... (thousands of attempts later) ...
    - Doa1
    - Dob1
    - Doc1
    - **Dog1** (Success!)

2. Dictionary Attack

A Dictionary Attack is a **targeted** method. Instead of trying every random combination, it uses a pre-defined list of likely passwords.

- **Logic:** Probability. It assumes that humans are predictable and use real words or common patterns.
- **The Procedure:**
    1. The attacker loads a "wordlist" (a text file containing millions of common passwords, leaked passwords from other breaches, and dictionary words).
    2. The software hashes the first word in the list (e.g., "admin").
    3. It compares that hash to the stolen database.

4. If it doesn't match, it moves to the next word (e.g., "qwerty").

- **Example:** A hacker uses the famous "RockYou.txt" wordlist (which contains over 14 million real-world passwords). If your password is iloveyou, the dictionary attack will find it in milliseconds, whereas a brute force attack might take hours to cycle through all the random letter combinations to get there.

Key Differences

Feature	Brute Force Attack	Dictionary Attack
Strategy	Tries every possible combination.	Tries a list of likely "human" words.
Success Rate	**100%** (given enough time).	High for common passwords; 0% for random strings.
Speed	Very slow for long passwords.	Very fast (skips the "nonsense" combinations).
Efficiency	Inefficient; wastes time on zQx9!#.	Highly efficient; focuses on Summer2024.

The "Hybrid" Attack (The Best of Both)

Modern hackers rarely use a "pure" dictionary attack. They use a **Hybrid Attack**, which takes a dictionary word and applies brute-force rules to it.[9]

Example:

- **Dictionary word:** monkey
- **Hybrid Rules:** Add a year at the end, capitalize the first letter, and swap "s" for "$".[10]
- **Results tried:** Monkey2024, M0nkey!, mOnkey123.

How to Defend Against Both

1. **Length is King:** Adding just two characters to a password can turn a Brute Force attack from taking **2 hours** to taking **2 decades**.
2. **Salting:** As we discussed earlier, salting makes Dictionary Attacks (and Rainbow Tables) useless because the hacker can't use a "pre-calculated" list.[11]
3. **Account Lockouts:** Most websites will lock your account after 5 or 10 failed tries. This completely kills Brute Force attacks, as the computer needs millions of tries to succeed.

6. Analyze why weak passwords fail.

Weak passwords fail because they are designed for **human convenience** rather than **machine complexity**. While a human sees a word like "Password123" as a combination of three different

elements (Word + Number + Symbol), a computer sees it as a predictable pattern that can be guessed in less than a second.

Here is a detailed analysis of why these passwords fail across four critical dimensions.

1. Low Entropy (Lack of Randomness)

Entropy is the measure of how unpredictable a password is. Weak passwords have low entropy because they follow common human language patterns.

- **Predictability:** Humans tend to use "anchor" points. We capitalize the first letter, use a common word, and put a number or "!" at the very end.
- **Mathematical Scale:** A 6-character password using only lowercase letters has 26^6 (about 308 million) combinations. A modern GPU can check these in **less than one millisecond**.

2. Vulnerability to Dictionary & Hybrid Attacks

Weak passwords usually consist of words found in the dictionary or names of popular culture items.

- **Wordlists:** Hackers use "RockYou.txt," a famous list of millions of real passwords leaked in past breaches. If your password is on that list, it doesn't matter how many symbols you added; the computer finds it instantly.
- **Rule-Based Attacks:** Hackers know people swap 'a' for '@' or 's' for '$'. Attack software automatically applies these "leetspeak" rules to every word in the dictionary.
    - *Example:* If a hacker tries the word "Admin," their software automatically tries @dmin, Adm1n!, and admin2024.

3. Credential Stuffing (The "Reused Password" Trap)

A weak password often fails not because it was guessed, but because it was **stolen from somewhere else**.

- **The Chain Reaction:** If you use the same weak password for your "Pizza Reward" account and your "Main Email," a breach at the pizza shop gives hackers the key to your entire digital life.
- **Automation:** Hackers use bots to take millions of leaked username/password pairs and "stuff" them into login pages for banks, social media, and retailers to see which ones work.[3]

4. Social Engineering & OSINT

Many weak passwords are based on **Personal Identifiable Information (PII)**. This makes them vulnerable to Open Source Intelligence (OSINT) gathering.

- **Public Knowledge:** If your password is your dog's name and your birth year (Rex1988), a hacker can find that information in seconds by looking at your Instagram or Facebook profile.

- **The "Secret Question" Loophole:** Weak passwords often go hand-in-hand with weak security questions (e.g., "What high school did you go to?"). These are easily bypassed by a quick search of your LinkedIn.

Summary of Failure Points

Reason for Failure	How Hackers Exploit It
Short Length	Brute force (trying every combo).
Common Words	Dictionary attacks.
Predictable Patterns	Hybrid attacks (adding '123' or '!').
Reuse	Credential stuffing from other breaches.
Personal Info	Social engineering/OSINT.

The "Cost of Cracking"

In 2026, the cost of computing power continues to drop. What used to take a week to crack ten years ago now takes a fraction of a second. **Complexity** is no longer enough; **Length** is the only thing that significantly increases the "work" a hacker's computer must do to fail.

Would you like me to show you how to create a "Passphrase" that is easy for you to remember but impossible for these attacks to crack?

7. Study MFA and its importance.

Multi-Factor Authentication (MFA) is the single most effective way to prevent unauthorized access to your accounts. In 2026, relying on a password alone—no matter how complex—is considered a major security risk.

MFA works on the principle of **layered defense**. If a hacker steals your password (the first layer), they are still blocked by the additional layers they do not have.

1. The Three Core Pillars of MFA

For a system to be true MFA, it must pull from at least two of these different categories:

Category	Definition	Examples
Something You Know	Knowledge you have stored in your brain.	Passwords, PINs, secret questions.
Something You Have	A physical or digital object in your possession.	Smartphone (app), Security Key (YubiKey), Smart Card.
Something You Are	Unique biological traits (Biometrics).	Fingerprint, FaceID, Retina scan.

Note: Entering a password and then a secret question is technically **Two-Step Verification**, not 2FA/MFA, because both factors are "Something You Know." True MFA requires different *types* of factors.

2. Why MFA is Essential in 2026

According to security research from Microsoft and Google, MFA can block over **99.9% of automated account-takeover attacks**.

It Stops the Most Common Attacks:

- **Credential Stuffing:** Even if a hacker finds your password in a leaked database from a different site, they cannot log in to your bank or email without your physical phone.
- **Phishing:** Even if you accidentally type your password into a fake website, the hacker can't get past the prompt for your fingerprint or hardware key.
- **Brute Force:** Since most MFA systems only allow a few attempts, a hacker cannot use a computer to guess your secondary code.

3. The "MFA Spectrum" (From Weakest to Strongest)

Not all MFA methods are equally secure. As hackers have evolved, some methods have become easier to bypass.

1. **SMS/Email Codes (Good):** Better than nothing, but vulnerable to "SIM Swapping" where a hacker steals your phone number.
2. **Authenticator Apps (Better):** Apps like Google Authenticator or Authy generate codes locally on your device, making them much harder to intercept.
3. **Push Notifications (Great):** You get a popup on your phone asking "Is this you?" with a Yes/No button. It's fast and secure.
4. **Hardware Keys & Passkeys (Best):** Using a physical USB key (FIDO2) or device-level biometrics. These are "Phishing-Resistant" because the security check is tied to the actual website URL—it won't trigger on a fake site.

4. The Move Toward "Passkeys"

In 2026, many companies are replacing passwords and MFA with **Passkeys**.

- Instead of a password + a code, your phone creates a unique cryptographic key for that specific website.
- You simply use your face or fingerprint to unlock the key.
- **Importance:** This removes the "human" element entirely—you can't "forget" a passkey, and a hacker can't "guess" it.

Summary: The Importance Checklist

- [ ] **Prevents Identity Theft:** Stops 99% of automated attacks.[17]
- [ ] **Compliance:** Many jobs and insurance policies now *require* MFA.

- [ ] **Peace of Mind:** You don't have to panic every time you hear about a password leak.

8. Write recommendations for strong authentication.

In 2026, the definition of "strong authentication" has shifted away from simply having a complex password. Modern standards, driven by **NIST (National Institute of Standards and Technology)** and the **FIDO Alliance**, focus on making authentication **phishing-resistant** and **context-aware**.

Here are the professional recommendations for implementing or practicing strong authentication today.

1. Adopt Phishing-Resistant MFA

Traditional MFA (like SMS codes or basic push notifications) can be intercepted or bypassed by "push bombing" attacks. Strong authentication now requires methods that are cryptographically bound to the website or service.

- **Passkeys (FIDO2/WebAuthn):** The gold standard for 2026. Instead of a password, your device creates a unique cryptographic key pair. You unlock it with biometrics (FaceID/Fingerprint). It is impossible to phish because the key only works on the legitimate website.
- **Hardware Security Keys:** Devices like a YubiKey or Google Titan Key. These require a physical touch to authorize a login, ensuring the user is physically present at the machine.
- **Certificate-Based Authentication (CBA):** Primarily for corporate environments, this installs a digital certificate on a specific device. Only that device can ever log in to the network.

2. Implement Adaptive (Risk-Based) Authentication

Strong authentication should not be a "one size fits all" process. Modern systems evaluate the **context** of the login before deciding how much proof to ask for.

- **Impossible Travel:** If you logged in from New York 10 minutes ago and someone tries to log in from London now, the system should block the attempt or require a high-assurance factor (like a hardware key).
- **Device Health Checks:** The system should verify the device is encrypted, has its firewall on, and is not "jailbroken" before allowing access to sensitive data.
- **Behavioral Biometrics:** Advanced systems monitor typing speed, mouse movement patterns, and gait (if on mobile) to ensure the person using the device hasn't changed since the initial login.

3. Modernize Password Policies (The NIST Standard)

If you must use passwords (for legacy systems), follow the updated **NIST SP 800-63B** guidelines. Most "old" rules actually make security worse.

- **Length > Complexity:** A 15-character passphrase (e.g., blue-skating-giraffe-rain) is significantly stronger than an 8-character complex password (e.g., P@ss12!).
- **Stop Periodic Resets:** Do not force users to change passwords every 90 days. This leads to "password cycling" (e.g., Summer2025!, Fall2025!) which hackers can easily guess. Only reset if there is evidence of a breach.
- **Ban Common Passwords:** Systems should check new passwords against a "blocklist" of known breached passwords (like 123456 or password).

4. Use Managed Identity (SSO & Password Managers)

To prevent "Credential Stuffing" (where a breach at one site affects all your others), you must ensure every account has a unique credential.

- **Single Sign-On (SSO):** Use a trusted Identity Provider (Okta, Microsoft Entra, Google Workspace) to log in to all your work apps. This centralizes security, so you only have to secure one "door" with the strongest possible MFA.
- **Enterprise Password Managers:** For accounts that don't support SSO, use a manager (Bitwarden, 1Password) to generate and store high-entropy, random strings.

Summary Recommendation Table

Security Level	Authentication Method	Recommended For
Basic	Password + SMS/Email OTP	Low-risk personal accounts (not ideal).
Advanced	Password + Authenticator App (TOTP)	General social media and personal banking.
Strong	**Passkeys** or **FIDO2 Hardware Keys**	Email, Finance, and Privileged Work accounts.
Enterprise	**Adaptive MFA + Zero Trust**	Corporate networks and high-value data.