Designing and Implementing Secure Authentication and Access Control in Cloud Environments

Cloud Authentication

1. Identity Provider (IAM / Azure AD)

In cloud environments, **Identity Providers (IdPs)** are responsible for managing user identities and verifying who is trying to access cloud resources.

- **AWS IAM** and **Azure Active Directory (Azure AD)** act as centralized identity management systems.
- They store user accounts, groups, roles, and credentials.
- When a user or service requests access, the Identity Provider authenticates the identity before allowing any interaction with cloud resources.

This is similar to how **Linux manages users and groups** using /etc/passwd, /etc/shadow, and /etc/group.

---

2. Password + Multi-Factor Authentication (MFA)

Authentication in the cloud often uses **multiple layers** of verification:

- **Password**: Something the user knows
- **MFA**: Something the user has (OTP app, SMS, hardware token)

MFA significantly improves security by preventing access even if a password is compromised.

This directly parallels Linux systems where:

- Password authentication is used for login
- Additional security layers (SSH keys, PAM modules, OTP) can be enforced

---

3. Token-Based Authentication

After successful authentication, cloud platforms issue **temporary security tokens**:

- Tokens prove identity without repeatedly sending passwords
- Tokens have **limited validity** and defined permissions
- Common in API access and service-to-service communication

Examples:

- AWS STS tokens
- OAuth / JWT tokens in Azure

This is similar to Linux session handling, where a user logs in once and receives a **session context** that defines their access rights until logout or expiry.

Cloud Authorization

1. Role-Based Access Control (RBAC) via IAM Roles

Authorization defines **what an authenticated user is allowed to do**.

- Cloud platforms use **roles** instead of permanent permissions
- Roles are attached to users, groups, or services
- Permissions are granted based on job function, not individual identity

This mirrors Linux RBAC concepts:

- Users are assigned to groups
- Permissions are inherited through group membership

---

2. Policies in JSON

Cloud permissions are defined using **policy documents**, usually written in JSON:

- Policies explicitly allow or deny actions
- They specify:
  - Actions (e.g., read, write, delete)

    - Resources (e.g., VM, storage bucket)
    - Conditions (time, IP, MFA status)

Example idea (conceptual):

- "Allow read access to storage, but deny delete"

This is comparable to Linux:

- File permission rules (rwx)
- ACLs and sudo policies that precisely define allowed actions

---

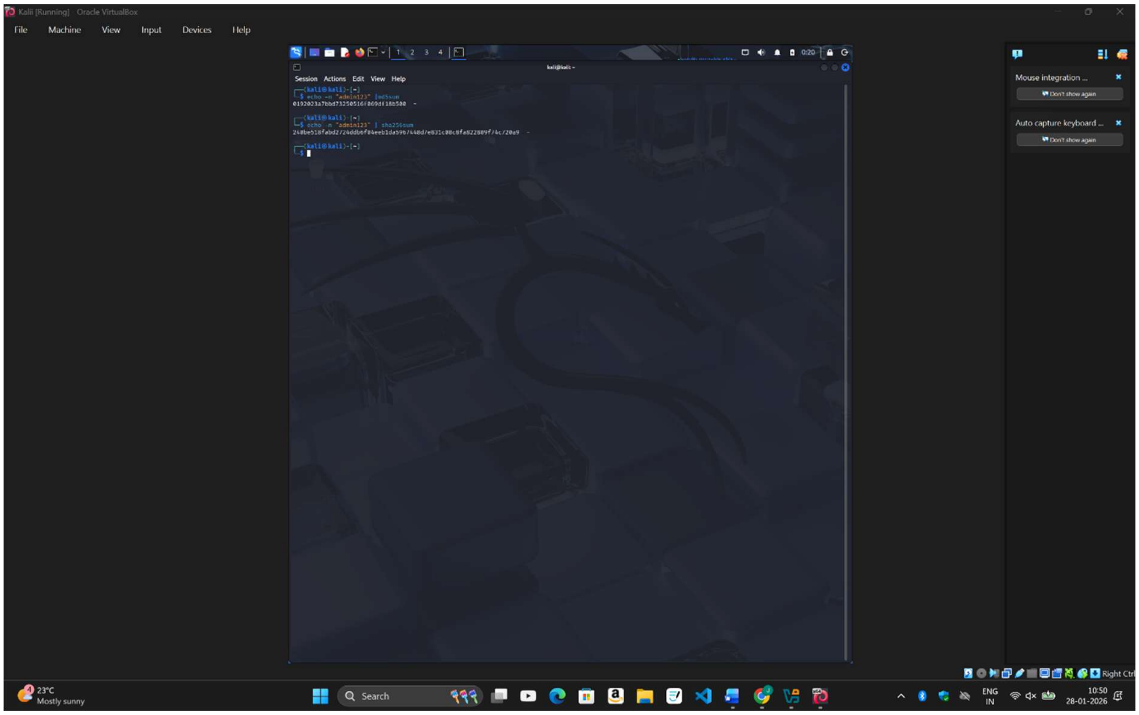
3. Principle of Least Privilege

The **least privilege principle** ensures that:

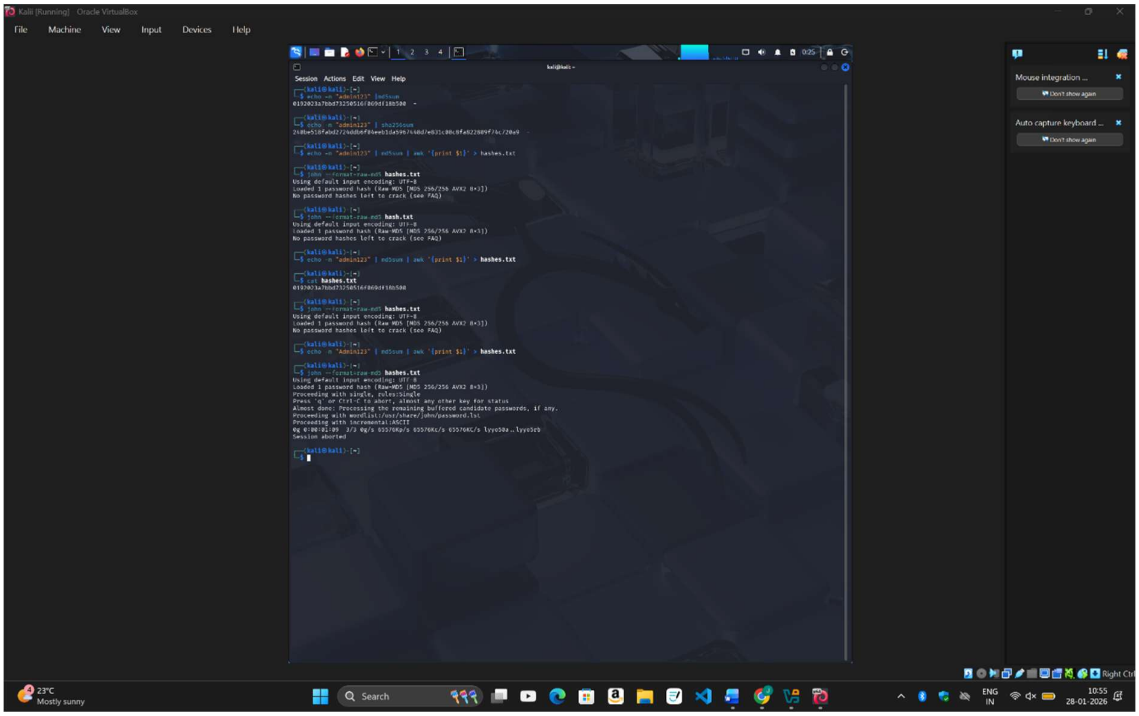
- Users and services receive **only the minimum permissions** required
- Reduces attack surface and accidental damage
- Permissions are reviewed and removed when no longer needed

Linux applies this principle by:

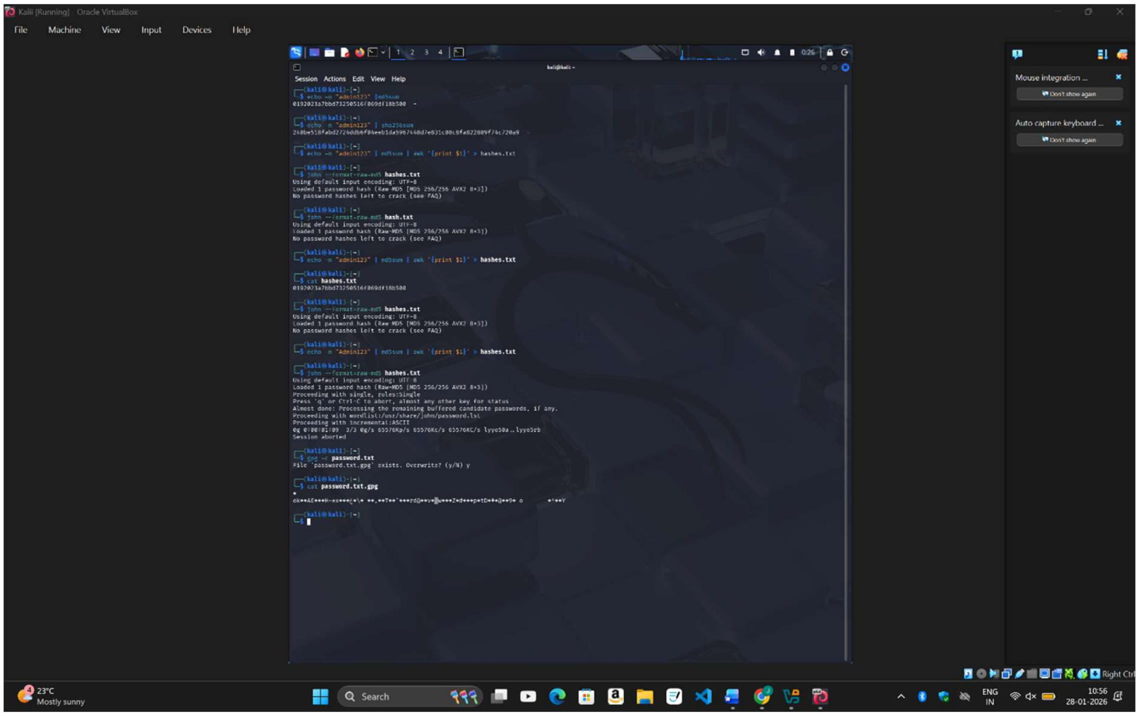
- Running services as non-root users
- Using sudo instead of full root access



```
┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" |md5sum
0192023a7bbd73250516f069df18b500  -

┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" | sha256sum
240be518fabd2724ddb6f04eeb1da5967448d7e831c08c8fa822809f74c720a9  -

┌──(kali㉿kali)-[~]
└─$ 
```



```
┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" |md5sum
0192023a7bbd73250516f069df18b500  -

┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" | sha256sum
240be518fabd2724ddb6f04eeb1da5967448d7e831c08c8fa822809f74c720a9  -

┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" | md5sum | awk '{print $1}' > hashes.txt

┌──(kali㉿kali)-[~]
└─$ john --format=raw-md5 hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
No password hashes left to crack (see FAQ)

┌──(kali㉿kali)-[~]
└─$ john --format=raw-md5 hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
No password hashes left to crack (see FAQ)

┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" | md5sum | awk '{print $1}' > hashes.txt

┌──(kali㉿kali)-[~]
└─$ cat hashes.txt
0192023a7bbd73250516f069df18b500

┌──(kali㉿kali)-[~]
└─$ john --format=raw-md5 hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
No password hashes left to crack (see FAQ)

┌──(kali㉿kali)-[~]
└─$ echo -n "Admin123" | md5sum | awk '{print $1}' > hashes.txt

┌──(kali㉿kali)-[~]
└─$ john --format=raw-md5 hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
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 0:00:01:09  3/3 0g/s 65576Kp/s 65576Kc/s 65576KC/s lyyo50a..lyyo5rb
Session aborted

┌──(kali㉿kali)-[~]
└─$ 
```



```
┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" |md5sum
0192023a7bbd73250516f069df18b500  -

┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" | sha256sum
240be518fabd2724ddb6f04eeb1da5967448d7e831c08c8fa822809f74c720a9  -

┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" | md5sum | awk '{print $1}' > hashes.txt

┌──(kali㉿kali)-[~]
└─$ john --format-raw-md5 hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
No password hashes left to crack (see FAQ)

┌──(kali㉿kali)-[~]
└─$ john --format-raw-md5 hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
No password hashes left to crack (see FAQ)

┌──(kali㉿kali)-[~]
└─$ echo -n "admin123" | md5sum | awk '{print $1}' > hashes.txt

┌──(kali㉿kali)-[~]
└─$ cat hashes.txt
0192023a7bbd73250516f069df18b500

┌──(kali㉿kali)-[~]
└─$ john --format-raw-md5 hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
No password hashes left to crack (see FAQ)

┌──(kali㉿kali)-[~]
└─$ echo -n "Admin123" | md5sum | awk '{print $1}' > hashes.txt

┌──(kali㉿kali)-[~]
└─$ john --format=raw-md5 hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
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 0:00:01:09  3/3 0g/s 65576Kp/s 65576Kc/s 65576KC/s lyyo50a..lyyo5rb
Session aborted

┌──(kali㉿kali)-[~]
└─$ gpg -c password.txt
File 'password.txt.gpg' exists. Overwrite? (y/N) y

┌──(kali㉿kali)-[~]
└─$ cat password.txt.gpg
♦
ok♦♦AE♦♦♦H-♦x♦♦♦¿♦\♦ ♦♦,♦♦T♦♦'♦♦♦rd@♦♦v♦▒w♦♦♦Zd♦♦♦p♦tD♦♦♦@♦♦9♦ o       ♦'♦♦Y

┌──(kali㉿kali)-[~]
└─$ █
```