Cognixia

Securing Your System

By - Ahmed

www.cognixia.com

1

Cognixia

Identity and Access Management (IAM)

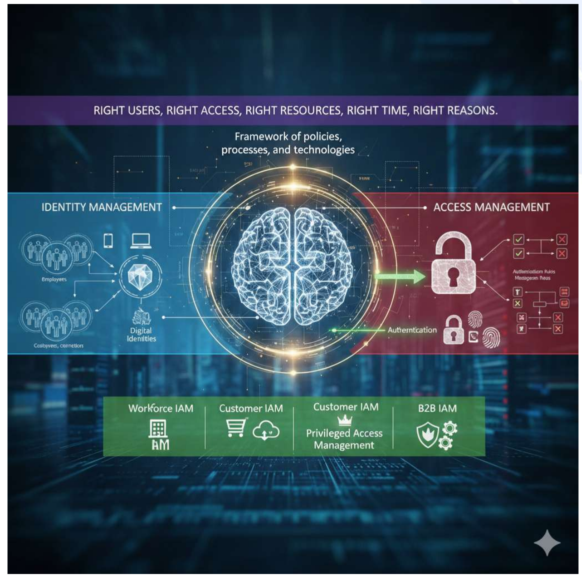
www.cognixia.com

2

What is IAM

Cognixia®

- Identity and Access Management (IAM) is a **framework** of **policies**, **processes**, and **technologies** that an **organization uses** to **manage digital identities** and **control user access** to its **resources**.
- Its core **purpose** is to **ensure** that the **right users** (or non-human entities like devices and software) **have** the **appropriate level** of **access** to the **right resources** at the **right time** and for the **right reasons**.
- IAM answers four key questions:
  - Who are you? → *Identity*
  - Can you prove it? → *Authentication*
  - What are you allowed to do? → *Authorization*
  - Should you still have that access? → *Governance & lifecycle*



www.cognixia.com

3

IAM | Authentication (AuthN)

Cognixia®

- Authentication is the process of **verifying a user's identity**.
- It **confirms** that the user (or entity) trying to access a system or resource is **actually who they claim to be**.
- Goal: To answer the question, "Are you who you say you are?"
- Method: The **user presents credentials** that are verified **against the system's records**.
- Outcome: **Success grants access** to the system. **Failure denies entry**.
- Common Authentication Factors:
  - **Authentication relies on factors** that prove identity, typically **using one or more of these types** (which leads to Multi-Factor Authentication):
    - Something you KNOW: **Password**, **PIN**, or **secret question**.
    - Something you HAVE: **Mobile phone** (*for an OTP/SMS code*), **security key** (*YubiKey*), or **smart card**.
    - Something you ARE: Biometric data like a **fingerprint**, **retina scan**, or **face recognition**.

www.cognixia.com

4

IAM | Authorization (AuthZ)

- Authorization is the process of **determining** what an **authenticated user is permitted to do**, what data **they can access**, and **what actions they can perform** within the system.
- Goal: To answer the question, **"What are you allowed to see or do?"**
- Method: The **system checks** the **user's roles**, **permissions**, or **access policies** against the requested resource.
- Outcome: **Grants or denies access** to a specific function or resource within the system.

Data Protection

Cognixia®

What is Data Protection

- Protecting data **involves** a comprehensive **set of strategies**, **processes**, and technologies designed **to secure information** from **unauthorized access**, corruption, or loss throughout its entire lifecycle.
- The **primary goal** is to ensure the **Confidentiality, Integrity, and Availability (CIA)** of the data.

www.cognixia.com

7

Cognixia®

Protecting Data at Rest (Stored)

- Data at rest **refers** to **inactive data stored physically** on any **medium** (*hard drives, databases, flash drives, or cloud storage*).
- Protection **focuses primarily** on **Confidentiality** and **restricting unauthorized physical** or **logical access**.
- **Encryption**
  - This is the most critical control.
  - **Data** must be **scrambled** so that **unauthorized** parties who gain access to the storage **cannot read** the information.
  - **Use Case**
    - ✓ A **healthcare provider encrypts patient health records** (PHR) stored in its database.
  - **Example**
    - ✓ A bank uses **Transparent Data Encryption (TDE)** within its core banking database (*e.g., Oracle or SQL Server*) to **encrypt financial account numbers** and transaction history, meeting **PCI DSS** (Payment Card Industry Data Security Standard) requirements.

www.cognixia.com

8

Protecting Data in Transit (Moving)

- Data in transit **refers** to **data actively moving** from **one location** to **another across** a **network** (*internet, VPN, or internal corporate network*).

- Protection focuses on **Confidentiality** and **Integrity** to **prevent eavesdropping** or **tampering during transmission**.

- **Encryption Protocols**
  - **Data** must be **encapsulated** within a **secure, encrypted tunnel** as it travels.
  - **Use Case**
    - A customer submits an order on an e-commerce website.
    - The **connection** between the **customer's browser** and the **web server** must be encrypted.
  - **Example**
    - ✓ All **communication between** a **user** and a **service provider** is mandated to use **TLS/SSL (HTTPS)** to ensure that login credentials and data payloads cannot be intercepted by Man-in-the-Middle attacks.

Protecting Data in Use (Processing)

- Data in use **refers** to **data** that is **actively being processed by a CPU** or **residing in volatile memory (RAM)**.

- This **state** is the **most vulnerable** because the **data must be** in its **decrypted form for computation**.

- **Trusted Execution Environments (TEEs)**
  - This is the leading **hardware-based strategy**.
  - **TEEs** create a **secure, isolated area of the processor** (an **enclave**) that **protects** the **data** and **code loaded inside** it, even from a compromised Operating System or Hypervisor.
  - **Use Case**
    - ✓ A **financial application** needs to **perform** a **highly sensitive credit check**.
    - ✓ The **application executes** the **check inside** a **TEE**, **ensuring** that the **customer's social security number** and the **scoring logic** remain invisible and tamper-proof to external processes.
  - **Example**
    - ✓ **Cloud providers** offer **Confidential Computing** services, allowing customers to run workloads on rented cloud hardware with a verifiable guarantee that the cloud operator itself cannot view the encrypted data while it's in use.

- **Homomorphic Encryption (HE)**
  - This is an **advanced cryptographic strategy** that **allows computation** to be **performed directly** on **encrypted data**.
  - **Use Case**
    - ✓ A research facility **wants** to **analyze** a **proprietary dataset** with a cloud service.
    - ✓ The **HE system allows** the cloud to **run statistical analysis** on the **data without ever seeing the raw values.**

Securing API(s)

11

Securing APIs

- **Securing** the **API** itself **involves defining strict rules** for how **request**s are **made**, **processed**, and **responded** to.
- The focus is on **implementing** strong **Confidentiality, Integrity, and Availability** controls.
- **Authentication and Identity Verification**
  - **APIs must confirm** the **identit**y of the **user** or client application **making** the **request.**
  - **OAuth 2.0**
    - It's used to **grant client** applications **limited access** to a user's resources on an API without exposing the user's credentials.
  - **OpenID Connect (OIDC)**
    - It **allows clients** to **verify** the **end-user's identity** and **obtain basic profile i**nformation using **ID Tokens (JWTs)**.
  - **API Keys (for application identity)**
    - These are simple, **unique identifiers** used for **basic client authentication** and **tracking**, but they should **never** be **used** for **identifying end-users** due to weak security controls.

12

Securing APIs ... continue

- **Authorization and Access Control**
  - **Once authenticated**, the API must **determine what** the **user** or application is **allowed to do**.
  - **Principle of Least Privilege**
    - Grant only the **minimum permissions necessary** for the specific task.
  - **Role-Based Access Control (RBAC)**
    - **Define roles** (*e.g., admin, read-only*) and check the **user's role** against the **required permissions** for the requested endpoint.
  - **Scope Checking**
    - For OAuth-based APIs, the **API must verify** that the **Access Token** has the **necessar**y **scopes** (*permissions*) to **perform** the **requested action** (*e.g., a token with a read:profile scope cannot perform a write:data operation*).
  - **Input Validation**
    - **Strictly validate** all **input data** (*parameters, headers, body*) **against expected formats**, **types**, and **lengths**.
    - This **prevents injection attacks** (*like SQL or XSS*).

www.cognixia.com

13

Securing APIs ... continue

- **Data Confidentiality and Integrity**
  - **Mandatory TLS/SSL**
    - All **API traffic** must be **secured** using **HTTPS/TLS** to ensure **encryption in transit**, preventing network eavesdropping (*Man-in-the-Middle attacks*).
  - **Content Encryption**
    - **Encrypt sensitive data payloads** (*fields like social security numbers*) **even *before* they** are **sent over TLS** for **end-to-end encryption**.
  - **Logging and Auditing**
    - **Log** all **API transactions**, especially **failed authentication attempts** and requests to sensitive endpoints.

www.cognixia.com

14

Securing API Gateways

Securing APIs Gateways

- An **API Gateway** is a **single-entry** point for **all API calls**, acting as a traffic cop and providing centralized security, management, and monitoring.

- Placing security controls at the Gateway is essential for defense in depth.

- **Centralized Policy Enforcement**
  - The **Gateway can enforce security policies** across **all APIs** without requiring individual developers to implement them repeatedly.

  - **Authentication Offloading**
    - ✓ The Gateway **can handle** the **initial authentication** (*validating the OAuth token or API key*) and then pass the verified identity to the backend service.
    - ✓ This **prevents** the **backend service** from having to **perform token validation**.

  - **Client Authorization**
    - ✓ It **checks** if the **client application** is even registered and **authorized** to **access** the **Gateway**.

Securing APIs Gateways … continue

- **Traffic Management and Threat Prevention**
  - These controls **protect** the **Availability** of the **API services** by managing incoming load.
  - **Rate Limiting**
    - **Imposing** a **limit** on the **number of requests** a **single user** or **application** can make in a given **time period** (*e.g., 100 requests per minute*).
    - This **prevents Denial of Service (DoS)** attacks and **ensures fair usage**.
  - **Throttling**
    - **Applying restrictions** based on **service capacity** (*slowing down requests when the backend is under load*) rather than just a fixed rate.
  - **Web Application Firewalls (WAF)**
    - **Integrating** a **WAF** to inspect API traffic for known attack patterns (*e.g., common exploits against OWASP Top 10 vulnerabilities*) and **block malicious requests** before they reach the backend services.
- **Cross-Origin Resource Sharing (CORS) Configuration**
  - When APIs are **consumed by front-end web applications hosted** on **different domains**, the Gateway must correctly **configure CORS headers** (*e.g., Access-Control-Allow-Origin*) to **prevent unauthorized cross-domain requests**, which can lead to security risks if misconfigured.

www.cognixia.com

17

Compliance-Driven Design

www.cognixia.com

18

What is Compliance-Driven Design

- Compliance-Driven Design (CDD) is a **methodology** where **security requirements and regulatory mandates** are **integrated** into the **earliest stages** of **system architecture**, **design**, and **development**, rather than **being added** as an **afterthought**.
- The core idea is to treat compliance not as a checklist for auditors, but as a set of non-functional requirements that fundamentally **drive design decisions** to ensure the system is secure and legally sound from inception.
- Compliance-Driven Design means **architecting systems, applications, and processes in a way that meets legal, industry, and organizational requirements**.
- This **ensures**:
  - Legal protection
  - Security consistency
  - Lower audit failures
  - Faster certifications
  - Reduced risk (financial, operational, reputational)
- Compliance becomes an architectural requirement just like performance, scalability, and reliability.

Regulatory Considerations

- The **primary driver** for **Compliance-Driven Design** is **compliance** with **external laws** and **regulations**.
- These **rules dictate** the necessary **security**, **privacy**, and **auditing controls** that must be **embedded i**nto the **system**.
  - **Data Privacy Regulations (GDPR, CCPA)**
    - These **laws focus** on **controlling** the **use** and **storage** of **Personal Data (PD)** and **granting rights** to the **consumer**.
    - **Design Mandate: Data Minimization**
      - ✓ **Design systems** to **collect** and **retain only** the **essential data** needed for a specific, stated purpose.
      - ✓ **Example:** A **sign-up process** should only **ask** for an **email address**, **postponing** the **collection** of a **mailing address until** the **user initiates** a **physical transaction**.
    - **Design Mandate: Privacy Controls**
      - ✓ **Build mechanisms** to **honor user rights automatically**.
      - ✓ **Example: Design** the **database** and **APIs** to quickly and **permanently execute** the **Right to Erasure** (*Right to be Forgotten*) **without impacting system stability**.
      - ✓ **Require auditable logs** for **every access to PD**.
    - **Design Mandate: Purpose Limitation**
      - ✓ **Structure** the **system** so that **data collected** for **one specific purpose** (e.g., billing) **cannot be** easily **used f**or an **unrelated purpose** (e.g., marketing) **without explicit consent**.

Regulatory Considerations … continue

- **Industry and Financial Regulations (HIPAA, PCI DSS)**
  - These rules dictate specific technical and procedural controls required for highly sensitive data types or business operations.
  - **Design Mandate: HIPAA Security Rule**
    - **Systems designed** for **healthcare** must **incorporate specific technical safeguards** for **electronic Protected Health Information (ePHI)**.
    - **Example**
      - ✓ The **logging system** must be designed to **capture all access** to **ePHI** and **retain** those **audit logs** for the **mandated regulatory period** (*e.g., six years*).
      - ✓ **All authentication requires** a strong, **recorded audit trail**.
  - **Design Mandate: PCI DSS (Payment Card Industry Data Security Standard)**
    - **Systems** that **store**, **process**, or **transmit credit card data** must be **designed** to **minimize** the **risk** of **card data compromise**.
    - **Example**
      - ✓ **Design** the payment module to use **Tokenization** instead of **storing** the **raw Primary Account Numbe**r (PAN).
      - ✓ The system architecture must **strictly segment** the **network** of the **payment module** from the **rest** of the **corporate network** (*Segmentation Control*).
  - **Design Mandate: Sarbanes-Oxley Act (SOX)**
    - For financial data, **design controls** are required to **ensure** the **Integrity** of financial reporting.
    - **Example**
      - ✓ **Implement** strict **Role-Based Access Control (RBAC)** around the systems that can modify general ledger data, ensuring that **no single person** can **unilaterally execute** and **approve** a **material financial transaction**.

21

22