1.4 Common Access Control Scenarios

1.4.1 Web Applications:

User login and session management:

Access to different pages or functionalities based on user role:

Data access control:

1.4.2 Operating Systems:

File permissions:

User accounts and privileges:

System calls and process access:

1.4.3 Databases:

User accounts and roles:

Table and column permissions:

Data manipulation operations:

1.4.4 APIs (Application Programming Interfaces):

API authentication:

API authorization:

Data filtering and masking:

1.4.5 Cloud Computing:

Identity and Access Management (IAM):

Access control for storage buckets:

Access control for virtual machines:

Access control for other cloud services:

1.4 Common Access Control Scenarios

1.4.1 Web Applications:

User login and session management: Controlling access to user accounts and maintaining user sessions securely.

Detailed Explanation:

Authentication to verify user identity (e.g., username/password, MFA).

Session management to track user activity and maintain login status (e.g., session IDs, cookies).

Protecting session IDs from theft or manipulation (e.g., HTTPS, secure cookies).

Vulnerabilities:

Session hijacking: Attackers steal a user's session ID to impersonate them.

Session fixation: Attackers force a user to use a specific session ID that they control.

Predictable session IDs: Weakly generated session IDs are easier for attackers to guess.

Access to different pages or functionalities based on user role: Restricting access to administrative functions, user profiles, or other resources based on user privileges.

Detailed Explanation:

Different user roles (e.g., administrator, editor, user) have different access levels.

Authorization mechanisms determine what actions each role is allowed to perform.

This is often implemented using RBAC.

Vulnerabilities:

Bypassing authorization checks: Attackers manipulate URLs or API calls to access restricted resources.

Accessing admin pages without proper credentials: Attackers find ways to reach administrative interfaces without logging in as an administrator.

Data access control: Controlling who can view, modify, or delete data within the application.

Examples:

Viewing or editing other users' profiles.

Accessing sensitive financial information.

Modifying application settings.

Vulnerabilities:

Insecure Direct Object References (IDOR): Attackers manipulate object IDs to access unauthorized data.

SQL injection: Attackers inject malicious SQL code to bypass access controls and access or modify data.

1.4.2 Operating Systems:

File permissions: Controlling who can read, write, or execute files.

Detailed Explanation:

File systems (e.g., NTFS, ext4) use permissions to control access to files and directories.

Permissions typically define who can read, write, and execute a file.

This is often implemented using DAC.

Vulnerabilities:

Incorrect permission settings: Files or directories are given overly permissive access rights.

Privilege escalation exploits: Attackers find ways to gain higher privileges than they should have, allowing them to bypass file permissions.

User accounts and privileges: Managing user accounts and assigning appropriate privileges.

Detailed Explanation:

Operating systems manage user accounts and assign privileges to control what users can do.

Privileges can include the ability to install software, modify system settings, or access specific resources.

This is related to RBAC and the principle of least privilege.

Vulnerabilities:

Weak password policies: Users are allowed to choose weak passwords, making accounts easier to compromise.

Privilege escalation exploits: Attackers find ways to gain higher privileges than they should have, allowing them to perform unauthorized actions.

System calls and process access: Controlling which processes can access system resources and perform specific operations.

Detailed Explanation:

Operating systems control access to system resources through system calls.

Access control mechanisms ensure that processes can only access the resources they need.

This is related to Mandatory Access Control (MAC) in some systems.

Vulnerabilities:

Buffer overflows: Attackers exploit memory management errors to execute arbitrary code with elevated privileges.

Race conditions: Attackers exploit timing vulnerabilities to perform unauthorized actions.

1.4.3 Databases:

User accounts and roles: Managing user accounts and assigning appropriate roles for database access.

Detailed Explanation:

Databases use user accounts and roles to control who can access the database.

Roles define sets of permissions that can be assigned to users.

This is an example of RBAC.

Vulnerabilities:

Weak password policies: Database user accounts have weak passwords.

SQL injection: Attackers inject malicious SQL code to bypass authentication/authorization and gain unauthorized access to the database.

Table and column permissions: Controlling which users or roles can access specific tables or columns within a database.

Detailed Explanation:

Databases allow you to control access to specific tables and columns.

This allows for fine-grained control over data access.

This is related to fine-grained access control.

Vulnerabilities:

SQL injection: Attackers bypass access controls and access unauthorized data.

Insufficient permission checking: Database queries don't properly check if the user has permission to access the requested data.

Data manipulation operations: Controlling which users or roles can perform operations like SELECT, INSERT, UPDATE, and DELETE.

Detailed Explanation:

Databases allow you to control who can perform data manipulation operations.

This ensures data integrity and prevents unauthorized modifications.

Vulnerabilities:

SQL injection: Attackers bypass access controls and perform unauthorized data manipulation.

Insufficient permission checking: Database operations don't properly check if the user has permission to perform the requested action.

1.4.4 APIs (Application Programming Interfaces):

API authentication: Verifying the identity of applications or users accessing the API.

Detailed Explanation:

APIs use authentication mechanisms to verify the identity of the client making the request.

Common methods include API keys, OAuth 2.0, and JWTs.

Vulnerabilities:

Weak API keys: API keys are easily guessable or compromised.

Missing authentication: APIs are not properly protected and allow anonymous access.

Insecure OAuth implementations: OAuth flows are not implemented securely, leading to vulnerabilities.

API authorization: Controlling access to different API resources or functionalities based on the authenticated entity.

Detailed Explanation:

APIs use authorization mechanisms to control what resources and functionalities the authenticated client can access.

This ensures that clients only have access to the data and functions they are authorized to use.

This is often implemented using RBAC or ABAC.

Vulnerabilities:

Missing authorization checks: APIs don't properly check if the client has permission to access the requested resource.

IDOR: Attackers manipulate API endpoints or parameters to access unauthorized resources.

Privilege escalation: Attackers find ways to gain higher privileges than they should have, allowing them to access restricted API functions.

Data filtering and masking: Restricting or obscuring sensitive data returned by the API.

Detailed Explanation:

APIs may need to filter or mask sensitive data based on the client's permissions.

This prevents unauthorized clients from accessing sensitive information.

Vulnerabilities:

Insufficient data filtering: APIs return more data than the client is authorized to see.

Incorrect masking: Sensitive data is not properly obscured.

1.4.5 Cloud Computing:

Identity and Access Management (IAM): Managing user identities and access privileges for cloud resources.

Detailed Explanation:

Cloud providers (e.g., AWS, Azure, GCP) use IAM services to control access to cloud resources.

IAM allows you to define policies that specify who can access which resources and what actions they can perform.

Vulnerabilities:

Overly permissive IAM policies: IAM policies grant excessive permissions, allowing unintended access.

Misconfigured roles: IAM roles are not properly configured, leading to security vulnerabilities.

Access control for storage buckets: Controlling who can access data stored in cloud storage services.

Detailed Explanation:

Cloud storage services use access control mechanisms to control who can access data stored in buckets or containers.

Permissions can be set at the bucket level or at the object level.

Vulnerabilities:

Publicly accessible buckets: Storage buckets are configured to allow public access, exposing data to the internet.

Misconfigured bucket policies: Bucket policies are not properly configured, granting unintended access.

Access control for virtual machines: Controlling who can access and manage virtual machines in the cloud.

Detailed Explanation:

Cloud providers use access control mechanisms to control who can access and manage virtual machines.

Access can be controlled through SSH keys, security groups, and IAM policies.

Vulnerabilities:

Weak SSH keys: SSH keys used to access virtual machines are weak or compromised.

Misconfigured security groups: Security groups allow unintended network access to virtual machines.

Access control for other cloud services: Controlling access to other cloud services like databases, message queues, and serverless functions.

Detailed Explanation:

Cloud providers use access control mechanisms to control access to other cloud services.

Each service may have its own specific access control mechanisms.

Vulnerabilities:

Service-specific misconfigurations: Access control settings for specific cloud services are not properly configured.

Overly permissive policies: Policies grant more access than necessary.