Identity and Access Management (IAM) & Role-Based Access Control (RBAC) - Highlevel Concepts and Primer

Table of Contents

Introduction

Role-Based Access Control (RBAC) is an enterprise security framework that manages digital resource access through role assignments. This model operates by associating permissions with functional roles rather than individual users, thereby establishing a structured approach to access management.

In an enterprise context, RBAC implements systematic access control through predefined role definitions (or even user defined). Each role encompasses a specific set of permissions aligned with job functions and responsibilities. For instance:

- Admin roles may receive comprehensive system access privileges
- HR/PeopleOps personnel obtain focused access to personnel management systems
- Front Office/Sales staff receive designated permissions for customer relationship platforms
- Typically employees are granted basic access to shared resources and their personal information

The primary advantage of RBAC lies in its centralised permission management architecture. Rather than administering individual user permissions, which is a process prone to inconsistency and administrative overhead, organisations insead establish standardised role `profiles`. When staff transitions do occur, access modification is accomplished through role reassignment rather than granular permission reconfiguration.

THerefore this framework delivers two critical organisational benefits:

1. Enhanced security through systematic access control and privilege limitation, and
2. Streamlined administration via centralised role management/systems

RBAC thus serves as a foundational element of any organisational security architecture, ensuring appropriate resource access while maintaining operational efficiency. Its' structured approach to permission management aligns with both security best practices and administrative pragmatism.

RBAC

Role-based access control allows organizations to improve their security posture and comply with security regulations.

IAM

Identity and Access Management (IAM) is a comprehensive framework that enables organizations to manage digital identities and their access to resources. IAM consists of two primary components:

- Identity Management: Handles user authentication and identity lifecycle
- Access Management: Controls what authenticated users can access (RBAC)

RBAC Definition

Role-Based Access Control (RBAC) is a security paradigm that restricts system access based on an individual's role within an organization. It operates on three key principles:

1. Role Assignment: Users are assigned to roles
2. Role Authorization: Roles are granted permissions
3. Permission Activation: Users acquire permissions through role assignment

One role-based access control example is a set of permissions that allow users to read, edit, or delete articles in a writing application. There are two roles, a Writer and a Reader, and their respective permission levels are presented in this truth table. Using this table, you can assign permissions to each user.

Permission/Role	Writer	Reader
Edit	Yes	No
Delete	Yes	No
Read	Yes	Yes

In some cases, organizations will grant different levels of permission to distinct roles, or their permission levels may overlap. In the above example, one role (the reader) is a subset of another role which has more permissions (the writer).

Attribute-Based Access Control (ABAC)

ABAC evaluates a set of rules and policies to manage access rights according to specific attributes, such as environmental, system, object, or user information. It applies boolean logic to grant or deny access to users based on a complex evaluation of atomic or set-valued attributes and the relationship between them.

In practical terms, this allows you to write rules in eXtensible Access Control Markup Language (XACML), using key-value pairs like Role=Manager and Category=Financial

RBAC vs ABAC

While RBAC relies on pre-defined roles, ABAC is more dynamic and uses relation-based access control. You can use RBAC to determine access controls with broad strokes, while ABAC offers more granularity. For example, an RBAC system grants access to all managers, but an ABAC policy will only grant access to managers that are in the financial department. ABAC executes a more complex search, which requires more processing power and time, so you should only resort to ABAC when RBAC is insufficient.

RBAC Architecture

Hierarchical Model

- Implements role inheritance in a typical IAM hierarchy
- Senior roles acquire permissions of junior roles
- Reduces administrative overhead
- Maps naturally to organizational structure

Core Components

1. **Users**: Individuals or service accounts requiring access
2. **Roles**: Collections of permissions
3. **Permissions**: Allowed operations on resources
4. **Sessions**: Temporary activation of roles
5. **Constraints**: Rules limiting role assignments

RBAC Implementation Advice

Implementing RBAC across the organization is complex and often results in pushback from various stakeholders.

Some concrete steps to help minimise this should include,

- A needs analysis to examine job functions
- Consider any regulatory or audit requirements and assess the current security posture of the organization
- Narrow scope initially to focus on systems or applications that store sensitive data

- The implementation phase needs addresses in stages
- First address a small/core set of users
- Also start with coarse grained access control before increasing granularity
- Gather extensive feedback from users and filter back into the planning of the next stage
- Generate the resultant details for each Component to implement

Implementation Components

Role Scope

- Defines boundaries of role authority
- Typically mapped to resource types
- Can be hierarchical (e.g., organization → project → resource)

Role Groups

- Collections of users sharing similar access needs
- Simplifies administration
- Supports temporary access management

Role Bindings

- Links between roles and identities/groups
- Can be scoped to specific resources
- Supports multiple bindings per identity

Scoped Tokens

- Time-limited access credentials
- Can be restricted to specific operations
- Support delegation without privilege escalation

User Scopes

- `user:full` : Complete access within user context
- `user:info` : Read-only profile access
- `user:list-apps` : Application visibility permissions

Authorization Workflow

Evaluation Process

The evaluation process can differ slightly between providers. So the Policy Evaluation with AWS may differ slightly from Policy Evaluation with GCP

The process would have these components.

1. Identity Resolution
    - Username validation
    - Group membership verification
    - Token validation
2. Action Analysis
    - Method/function mapping
    - API endpoint resolution

- Verbs translation (HTTP methods)

3. Resource Context
    - Project/namespace identification
    - Resource type determination
    - Hierarchical position
4. Permission Evaluation
    - Policy aggregation
    - Role resolution
    - Constraint checking
    - Default deny principle application

Policy Types

1. **Identity-based Policies**
    - Attached to users/groups
    - Define allowed operations
    - Support conditions
2. **Resource-based Policies**
    - Attached to resources
    - Control access to specific items
    - Support cross-account access
3. **Service Control Policies (SCPs)**

    SCPs are more an AWS concept. They,

    - Define permission boundaries
    - Apply to organizational units
    - Override other policy types
4. **Access Control Lists (ACLs)**

    An access control list (ACL) is a table listing the permissions attached to computing resources. It tells the system which users can access an object, and which actions they can carry out. There is an entry for each user, which is linked to the security attributes of each object. ACL is most commonly used for traditional DAC systems but it is also used by various Cloud providers. Uses include,

    - Legacy access control
    - Simple allow/deny rules
    - Resource-specific permissions

Security Best Practices

Organizational Alignment

1. Mirror organizational structure in IAM hierarchy
2. Group resources by trust boundaries
3. Implement consistent naming conventions
4. Document role definitions and assignments

Least Privilege

1. Grant minimum necessary permissions

2. Use time-bound access when possible
3. Regularly review and revoke unused permissions
4. Implement just-in-time access like
5. Allow User Temporary Elevated Access

Administrative Controls

1. Use groups for role assignment
2. Implement approval workflows
3. Audit access patterns
4. Monitor privilege escalation
5. Implement break-glass procedures

Technical Controls

1. Enforce TLS for all communications
2. Implement session management
3. Follow token best practices like secure token storage
4. Enable comprehensive logging
5. Implement rate limiting

Attack Surface Analysis

Attack Surface Components

1. **Entry Points**
    - API endpoints
    - User interfaces
    - Service integrations
    - Authentication mechanisms
2. **Exit Points**
    - Data exports
    - System responses
    - Logging outputs
    - Integration callbacks
3. **Trust Boundaries**
    - Service perimeters
    - Network segments
    - Authentication boundaries
    - Authorization zones

Assessment Methodology

1. **Mapping**
    - Document all interfaces
    - Identify data flows
    - Map trust relationships
    - Catalog sensitive data
2. **Analysis**

- Risk assessment
- Threat modeling
- Vulnerability scanning
- Penetration testing

3. **Monitoring**
   - Access pattern analysis
   - Anomaly detection
   - Security event logging
   - Compliance auditing

Infrastructure Hardening

Platform Security

1. **Network Controls**
   - Implement network segmentation
   - Configure firewalls
   - Enable DDoS protection
   - Monitor traffic patterns
2. **System Hardening**
   - Disable unnecessary services
   - Apply security patches
   - Configure secure defaults
   - Implement endpoint protection
3. **Operational Security**
   - Implement change management
   - Monitor system health
   - Backup critical data
   - Plan disaster recovery

Service-Specific Hardening

1. **Authentication Services**
   - Implement MFA
   - Secure credential storage
   - Monitor authentication attempts
   - Implement account lockouts
2. **API Security**
   - Implement rate limiting
   - Validate input
   - Encrypt sensitive data
   - Monitor API usage
3. **Storage Security**
   - Encrypt data at rest
   - Implement access controls
   - Secure backup processes

- Monitor data access

Compliance Considerations

- GDPR requirements
- PDPA compliance - Country Specific PDPA Singapore
- Industry-specific regulations
- Regional requirements

Tools and Resources

1. **Security Testing**
   - OWASP ASD
   - ... freshen these
2. **Monitoring**
   - Audit logging
   - Security information and event management (SIEM)
   - Intrusion detection systems
   - Access analytics
3. **Documentation**
   - Security policies
   - Incident response plans
   - Compliance documentation
   - Training materials
   -

Implementation Patterns for Specific Use Cases

Microservices Architecture

1. **Service-to-Service Authentication**
   - JWT-based token authentication JWT RFC7519
   - Mutual TLS (mTLS) implementation
   - Service mesh integration

```yaml
# Example service mesh RBAC configuration
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: service-rbac
spec:
  selector:
    matchLabels:
      app: backend-service
  rules:
  - from:
    - source:
        principals: ["cluster.local/ns/default/sa/frontend-service"]
  to:
    - operation:
        methods: ["GET"]
        paths: ["/api/v1/*"]
```

2. **API Gateway Integration**

- Rate limiting per role
- Quota management
- API key rotation

```
{
  "apis": [
    {
      "path": "/api/v1/users",
      "roles": ["admin", "user-manager"],
      "rate_limit": {
        "admin": 1000,
        "user-manager": 100
      },
      "quota": {
        "admin": "unlimited",
        "user-manager": "10000/day"
      }
    }
  ]
}
```

Multi-tenant Applications

1. **Tenant Isolation**
   - Separate role hierarchies per tenant
   - Cross-tenant access controls
   - Resource namespace separation

```
-- Example tenant isolation schema
CREATE TABLE tenant_roles (
  tenant_id UUID,
  role_id UUID,
  role_name VARCHAR(50),
  parent_role_id UUID,
  CONSTRAINT pk_tenant_roles PRIMARY KEY (tenant_id, role_id)
);
```

2. **Resource Sharing**
   - Shared resource access patterns
   - Tenant-specific permissions
   - Resource ownership tracking

```
{
  "resource": "document-123",
  "sharing": {
    "type": "cross-tenant",
    "permissions": ["read", "comment"],
    "tenant_access": [
      {
        "tenant_id": "tenant-456",
        "roles": ["viewer"],
        "expires": "2024-12-31"
      }
    ]
  }
}
```

Cloud-Native Deployments

1. **Multi-Cloud IAM**
   - Cloud provider IAM integration

- Federation services
- Cross-cloud role mapping

```
# Terraform example of cross-cloud IAM
resource "aws_iam_role" "cross_cloud_role" {
  name = "cross-cloud-access"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Federated = "accounts.google.com"
        }
      }
    ]
  })
}
```

2. **Kubernetes Integration**

- RBAC for pod security
- Service account management
- Custom resource definitions

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

Detailed Attack Surface Analysis Methodologies

Entry Point Mapping

1. **Automated Discovery**

```
# Example Nmap scanning script
#!/bin/bash
nmap -sS -sV -A \
  --script=vuln \
  --script=auth \
  --script=discovery \
  -oX scan_results.xml \
  target_host
```

2. **API Security Analysis**

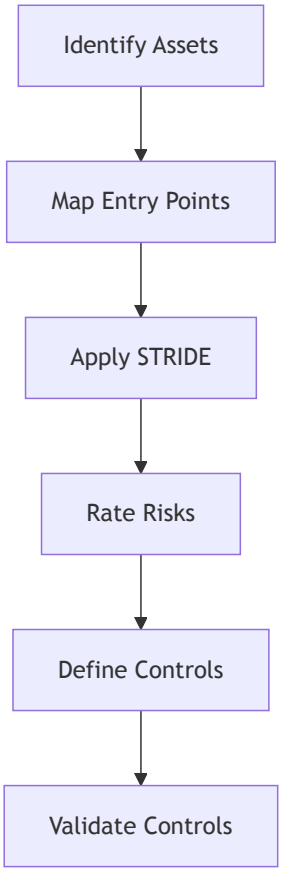
- OpenAPI/Swagger scanning
- Authentication bypass testing
- Authorization testing matrix

```
def test_api_auth_matrix(endpoints, roles):
    results = []
    for endpoint in endpoints:
        for role in roles:
            response = test_access(endpoint, role)
            results.append({
                'endpoint': endpoint,
                'role': role,
```

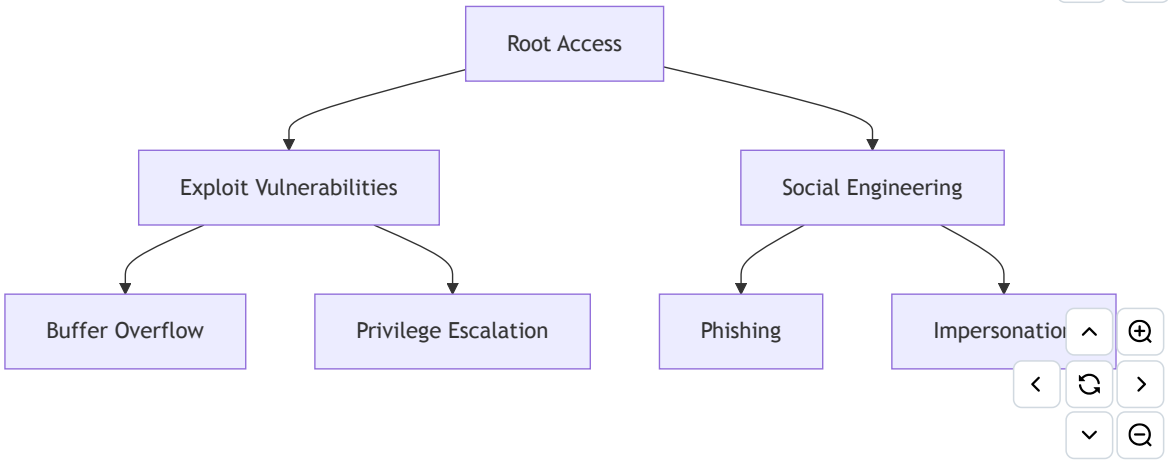
```
                'status': response.status_code,
                'expected': get_expected_access(endpoint, role)
            })
    return results
```

Threat Modeling Process

1. Stride Analysis
   - Spoofing
   - Tampering
   - Repudiation
   - Information Disclosure
   - Denial of Service
   - Elevation of Privilege



2. Attack Trees



Continuous Security Assessment

1. **Automated Scanning**

```yaml
# Example GitLab CI security scanning
security_scan:
  stage: test
  script:
    - safety check
    - bandit -r ./src
    - owasp-dependency-check
  artifacts:
    reports:
      security: gl-security-report.json
```

2. **Penetration Testing Framework**

```python
class SecurityTest:
    def __init__(self, target):
        self.target = target
        self.vulnerabilities = []

    def run_security_checks(self):
        self.test_authentication()
        self.test_authorization()
        self.test_input_validation()
        self.test_encryption()

    def report_findings(self):
        return {
            'target': self.target,
            'vulnerabilities': self.vulnerabilities,
            'risk_score': self.calculate_risk_score()
        }
```

Compliance Mapping for Specific Regulations

GDPR Compliance Matrix

GDPR Article	RBAC Control	Implementation
Art. 25	Privacy by Design	Role-based data access
Art. 32	Security Controls	Encryption, logging
Art. 35	DPIA	Risk assessment

[PCI-DSS] Requirements

Requirement	Control	Implementation
7.1	Role Definition	Least privilege
7.2	Access Control	Need-to-know
10.1	Audit Trails	Logging system

SOC2 Controls

```yaml
# Example SOC 2 control mapping
controls:
  access_control:
    type: "SOC 2 CC6.1"
    implementation:
      - Role-based access control
      - Multi-factor authentication
      - Access review process
    monitoring:
      - Access logs
      - Failed login attempts
      - Privilege changes
```

HIPAA Security Rule

```json
{
  "security_rule": {
    "administrative_safeguards": {
      "access_control": {
        "required": true,
        "controls": [
          "Role-based access",
          "Unique user identification",
          "Emergency access procedure"
        ]
      }
    }
  }
}
```

Infrastructure Hardening Guidelines

Network Security

1. **Segmentation**

```terraform
# Terraform network segmentation
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "production"
    Environment = "prod"
  }
}

resource "aws_subnet" "private" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"
}
```

2. **Firewall Rules**

```
# Example iptables configuration
iptables -A INPUT -p tcp --dport 22 -s 10.0.0.0/8 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -P INPUT DROP
```

System Hardening

1. **OS Hardening**

```
# Example Ansible playbook
- name: Harden OS
  hosts: all
  tasks:
    - name: Set password policy
      pam_pwquality:
        minlen: 14
        minclass: 4
        enforce_for_root: true

    - name: Configure audit logging
      template:
        src: audit.rules.j2
        dest: /etc/audit/rules.d/audit.rules
```

2. **Service Hardening**

```
# Example Nginx hardening
server {
  listen 443 ssl http2;
  ssl_protocols TLSv1.2 TLSv1.3;
  ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256;
  ssl_prefer_server_ciphers off;
  add_header Strict-Transport-Security "max-age=63072000" always;
}
```

Container Security

1. **Docker Security**

```
# Secure Dockerfile example
FROM alpine:3.14
RUN addgroup -S appgroup && adduser -S appuser -G appgroup
USER appuser
COPY --chown=appuser:appgroup ./app /app
EXPOSE 8080
CMD ["./app"]
```

2. **Kubernetes Security**

```
# Pod Security Policy
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
spec:
  privileged: false
  seLinux:
    rule: RunAsAny
  runAsUser:
    rule: MustRunAsNonRoot
  fsGroup:
```

```
    rule: RunAsAny
  volumes:
  - 'configMap'
  - 'emptyDir'
  - 'projected'
  - 'secret'
  - 'downwardAPI'
```

Monitoring and Detection

1. **Log Management**

```
# Example Filebeat configuration
filebeat.inputs:
- type: log
  enabled: true
  paths:
    - /var/log/auth.log
    - /var/log/syslog
  fields:
    type: system
    environment: production

output.elasticsearch:
  hosts: ["elasticsearch:9200"]
  index: "logs-%{[fields.type]}-%{+yyyy.MM.dd}"
```

2. **Intrusion Detection**

```
# Example Falco rules
- rule: Unauthorized Process
  desc: Detect unauthorized process execution
  condition: spawned_process and not proc.name in (allowed_processes)
  output: "Unauthorized process started (user=%user.name command=%proc.cmdline)"
  priority: WARNING
```

NIST RBAC

RBAC

NIST RBAC Model

Glossary

Identity Management

Access Management

RFC7519

JWT

mTLS

IAM

Google Resource Hierarchy

IAM Deepdive AWS

VERBS

Policy Evaluation GCP