4. MongoDB Self-Managed Security

- Authentication & Authorization
- Role-Based Access Control (RBAC)
- TLS/SSL configuration
- Network security
- Data at rest encryption
- Lab: Create users and enforce roles

Goal: secure MongoDB end-to-end with:

- Authentication & Authorization
- RBAC (roles)
- TLS/SSL
- Network security
- Data-at-rest encryption
- Hands-on user/role lab

---

0. Lab Environment (Common for All)

We'll assume a database called:

```
csc_compliance
```

Collections:

- entities
- filings
- reminders

Admin user we'll create: cscSecurityAdmin

Application user: cscAppUser

---

1. Authentication & Authorization (Enable & Test)

1.1 Concepts (Explain Before Lab)

- **Authentication** – “Who are you?” → users & passwords, x.509, etc.
- **Authorization** – “What can you do?” → roles & privileges.
- **RBAC** – Role-Based Access Control; roles define actions on resources.

1.2 Step 1 – Start MongoDB without auth (bootstrap admin)

You normally do this once on a new instance to create the first admin.

Linux / Mac

```
mongod --dbpath /data/db --bind_ip 127.0.0.1
```

(or use your existing config/service if already running without auth.)

Windows (cmd/PowerShell)

```
mongod --dbpath "C:\data\db" --bind_ip 127.0.0.1
```

For services, you may temporarily edit mongod.cfg to **comment out** security.authorization for bootstrap if needed.

1.3 Step 2 – Create the First Admin User

Connect:

```
mongosh "mongodb://127.0.0.1:27017"
```

Create admin user in admin DB:

```
use admin

db.createUser({
  user: "cscSecurityAdmin",
  pwd:  "StrongPassword!123",
  roles: [ { role: "userAdminAnyDatabase", db: "admin" },
           { role: "dbAdminAnyDatabase", db: "admin" },
           { role: "readWriteAnyDatabase", db: "admin" } ]
})
```

CSC Use Case

Bangalore security team wants a **central security admin** who can create/manage users across all CSC databases.

1.4 Step 3 – Enable Authorization in Config

Now we turn on **auth** properly.

Linux

Edit /etc/mongod.conf:

```
security:
  authorization: enabled
```

Restart:

```
sudo systemctl restart mongod
```

Windows (

C:\Program Files\MongoDB\Server\X.Y\bin\mongod.cfg

)

```
security:
  authorization: enabled
```

Restart service:

```
net stop MongoDB
net start MongoDB
```

Mac (Homebrew cfg, typically

/usr/local/etc/mongod.conf

or

/opt/homebrew/etc/mongod.conf

)

```
security:
  authorization: enabled
```

Restart:

```
brew services restart mongodb-community
```

1.5 Step 4 – Test Authentication

Now **unauthenticated** access should be denied.

```
mongosh "mongodb://127.0.0.1:27017"
```

Try:

```
show dbs   // should fail with unauthorized
```

Now authenticate:

```
use admin
db.auth("cscSecurityAdmin", "StrongPassword!123")

show dbs  // should now work
```

✅ CSC Use Case

No one (including internal tools) can now access CSC filings data without a valid account. Even Mac dev boxes for Mumbai engineers are properly locked down.

---

2. Role-Based Access Control (RBAC) – Create Application Users

2.1 Step 1 – Create Business DB & Sample Data

Still authenticated as cscSecurityAdmin:

```
use csc_compliance

db.filings.insertOne({
  entity_id: 1001,
  filing_type: "Annual Report",
  state: "DE",
  status: "OPEN",
  amount: 5000
})
```

2.2 Step 2 – Create an App User with Read/Write on One DB

In admin:

```
use admin

db.createUser({
  user: "cscAppUser",
  pwd:  "CSC-App-Password@2025",
  roles: [
    { role: "readWrite", db: "csc_compliance" }
  ]
})
```

Now test as app user:

```
mongosh "mongodb://cscAppUser:CSC-App-
Password@2025@127.0.0.1:27017/csc_compliance?authSource=admin"
```

Check:

```
db.filings.find()
db.filings.insertOne({ entity_id: 1002, filing_type: "Franchise Tax",
state: "CA", status: "OPEN", amount: 8000 })
```

Try to access another DB:

```
use admin
show collections  // should be unauthorized
```

✅ CSC Use Case

The **Compliance microservice** only needs readWrite on csc_compliance.
RBAC ensures it cannot touch other internal databases.

2.3 Lab: Create Read-Only Reporting User (Mumbai team)

Create a read-only user for BI/reporting tools:

```
use admin

db.createUser({
  user: "cscReportingUser",
  pwd:  "CSC-Report-Only@2025",
  roles: [
    { role: "read", db: "csc_compliance" }
  ]
})
```

Test:

```
mongosh "mongodb://cscReportingUser:CSC-Report-
Only@2025@127.0.0.1:27017/csc_compliance?authSource=admin"
```

Try:

```
db.filings.find().limit(2)         // OK
db.filings.insertOne({ ... })      // Should fail with unauthorized
```

✅ CSC Use Case

Mumbai analytics team can read filings safely without any risk of modifying or deleting production data.

---

3. TLS/SSL Configuration (Secure in Transit)

Do this on **Linux/Windows/Mac self-managed**. In Atlas it's handled for you (TLS is on by default).

3.1 Generate Self-Signed Cert (Lab Only)

On Linux/Mac (for lab):

```
mkdir -p ~/csc-certs
cd ~/csc-certs

openssl req -newkey rsa:4096 -new -x509 -days 365 -nodes \
  -out csc-mongo.pem -keyout csc-mongo.pem \
  -subj "/C=IN/ST=Maharashtra/L=Pune/O=CSC/OU=IT/CN=localhost"
```

Set permissions:

```
chmod 600 csc-mongo.pem
```

On Windows, you can generate using OpenSSL for Windows or a PKI tool and place csc-mongo.pem in e.g. C:\csc-certs\csc-mongo.pem.

3.2 Configure mongod to Use TLS

Linux (

/etc/mongod.conf

):

```
net:
  port: 27017
  bindIp: 0.0.0.0
  tls:
    mode: requireTLS
    certificateKeyFile: /home/<user>/csc-certs/csc-mongo.pem
```

Restart:

```
sudo systemctl restart mongod
```

Windows (

mongod.cfg

):

```
net:
  port: 27017
  bindIp: 0.0.0.0
  tls:
    mode: requireTLS
    certificateKeyFile: C:\csc-certs\csc-mongo.pem
```

Restart:

```
net stop MongoDB
net start MongoDB
```

Mac:

Similar to Linux, config path like /usr/local/etc/mongod.conf or /opt/homebrew/etc/mongod.conf.

3.3 Connect with TLS

```
mongosh "mongodb://cscAppUser:CSC-App-
Password@2025@localhost:27017/csc_compliance?authSource=admin&tls=true" \
  --tlsAllowInvalidCertificates
```

(For lab we allow invalid certs; in production you'd use proper CA-signed certs.)

✅ **CSC Use Case**

Connections from app servers in Bangalore/Mumbai to the DB are now **encrypted**, protecting client data and filings from network sniffing.

---

4. Network Security

4.1 Bind IP – Don't Listen on Everywhere (unless needed)

In mongod.conf:

```
net:
  bindIp: 127.0.0.1,10.0.0.10   # Only internal IPs, not public
```

- For dev (Mac/Windows) → often just 127.0.0.1.
- For prod (Linux) → internal VPC subnet, NOT 0.0.0.0 exposed to internet.

Reload service after this change.

✅ **CSC Use Case**

Only **app servers and bastion hosts** in CSC's private network can reach MongoDB. Externally, ports are blocked by firewall + security groups.

4.2 Firewalls / Security Groups

- **Linux**: use ufw / firewalld to allow only app servers' IPs.
- **Windows**: Windows Defender Firewall rules.
- **Atlas**: IP allow-lists in Atlas UI (Network Access tab).

Example (Linux ufw):

```
sudo ufw allow from 10.0.1.50 to any port 27017 proto tcp      # app server
sudo ufw deny 27017/tcp
```

✅ **CSC Use Case**

Citizen laptop on the internet cannot reach MongoDB; only CSC corporate systems in the right VLAN can.

---

5. Data-at-Rest Encryption (High-Level Steps)

5.1 Self-Managed: File System / Disk Encryption

Options:

- Linux: **LUKS**, encrypted volume, or cloud provider disk encryption (EBS, Azure Disk, etc.).
- Windows: **BitLocker**.
- Mac: **FileVault**.

MongoDB Enterprise also supports **Encrypted Storage Engine** with key management, but at minimum:

Put /data/db on an encrypted volume.

✅ **CSC Use Case**

If someone steals a hard drive from the Bangalore data center or a lost MacBook, raw MongoDB files are not readable without encryption keys.

5.2 Atlas: Encryption at Rest

Atlas provides:

- **Cloud provider encryption** (default for many tiers).
- Optional **Customer KMS** integration (AWS KMS, Azure Key Vault, GCP KMS).

Admins configure this via **Atlas Project → Security → Encryption at Rest**.

---

6. Atlas Security Mapping (Quick Overview)

Atlas (managed):

- **Authentication**: DB users created in Atlas UI or via API.
- **RBAC**: same MongoDB roles; plus Atlas project/org roles for admin access.
- **TLS**: always on for cluster endpoints.
- **Network Security**:
    - IP Access List (allow specific public IPs / ranges).
    - VPC Peering / PrivateLink for private connectivity.
- **Encryption at Rest**: via cloud + KMS.

✅ CSC Use Case

For new greenfield apps, CSC prefers Atlas so security controls (TLS, backups, encryption) are mostly **click-configured** instead of manually managed.

---

7. LAB – Users & Enforcement (90-Minute Practical Flow)

This is how you can run the 1.5-hour session.

Phase 1 – Bootstrap & Auth (25 min)

1. Start MongoDB without auth (local Linux/Mac/Windows).
2. Create cscSecurityAdmin in admin.
3. Enable authorization: enabled in config and restart.
4. Confirm unauthenticated access fails; admin auth works.

Phase 2 – RBAC & App Users (20 min)

1. Create csc_compliance DB and filings collection.
2. Create cscAppUser with readWrite on csc_compliance.
3. Create cscReportingUser with read only.
4. Verify:
   - App user can insert/update in csc_compliance.
   - Reporting user can read but not write.

Phase 3 – TLS & Network (25 min)

1. Generate a self-signed cert (Linux/Mac; on Windows pre-generated file provided).
2. Configure mongod to requireTLS.
3. Connect with mongosh using &tls=true and --tlsAllowInvalidCertificates.
4. Restrict bindIp to loopback + internal IP.
5. Discuss firewall rules; show one example (ufw or Atlas IP allow-list screenshot).

Phase 4 – Data at Rest & Atlas Overview (20 min)

1. Explain/discuss:
   - Putting /data/db on encrypted disk volumes (LUKS/BitLocker/FileVault).
   - Atlas encryption at rest & KMS integration.
2. In Atlas:
   - Show where to create DB users.
   - Show IP Access List.
   - Show that TLS is enforced.

---

8. Quick Security Checklist for CSC MongoDB Admins

- **Auth enabled** (authorization: enabled)
- **Admin user** exists and is stored securely
- **App users** have least-privilege roles (e.g., readWrite on specific DB only)
- **Reporting users** are read only
- **TLS** enabled; using valid certs in production
- **Network access restricted** (bindIp, firewall, Atlas IP allow list / VPC)
- **Data directory on encrypted disk** (or storage engine encryption in Enterprise)
- **Strong passwords** / secrets managed via vault (not in code)
