PERCONA

MongoDB Security: A Practical Guide

David Murphy
MongoDB Practice Manager
@ Percona

Agenda

- What do people typically get wrong with Mongo?
- What are the main security concepts?
- Using SSO via LDAP/SASL
- Mongo users and roles
  - How roles and resources work
  - Custom roles
- Setting up SSL
  - Custom CA’s
  - Invalid hostnames
- Disk encryption options
- Tracking actions in Mongo with auditing
- Reducing your attack surface with network planning

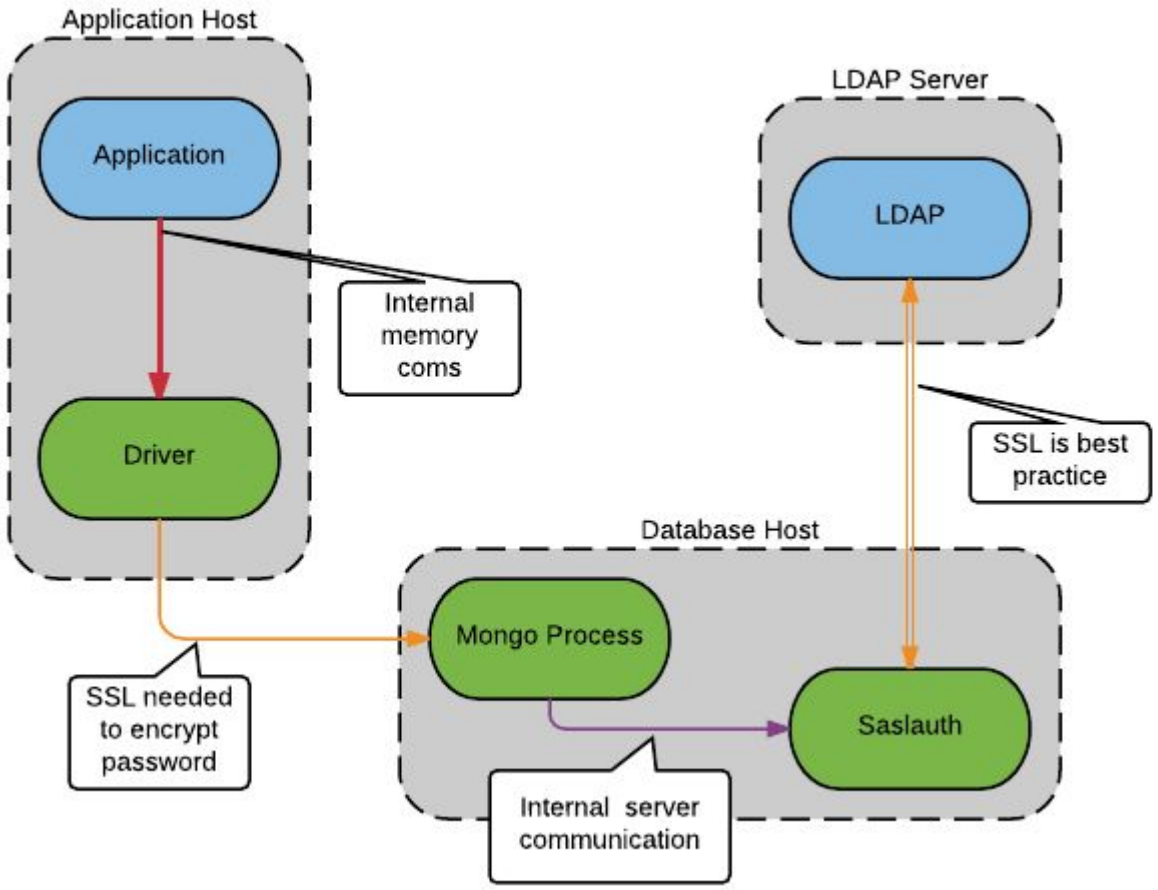
What do people get wrong?

- Not limiting which network is listening
- Using the default ports
- Not enabling authentication immediately
- When using auth, giving everyone broad access
- Not using LDAP to force password rotations
- Not forcing SSL usage on the database
- Not limiting DB access to known network devices (apphosts, load balancers)

Core areas of security

- **Authentication**. LDAP Authentication centralizes things in your company directory (for PCI).
- **Authorization**. What role-based access controls the database provides.
- **Encryption**. Broken into “At-Rest” and “In-Transit” as part of regular PCI requirements.
- **Governance**. Document validation and checking for sensitive data (such as SSN or birth data).
- **Auditing**. The ability to see who did what in the database (also required for PCI).

LDAP Authentication Flow



Steps in SASL Auth

- Application tells driver to use $external for auth
- Driver sends auth command to database
- Plaintext if not using SSL
- Database checks if auth and sasldpath is set
- Database passes user/pass to sasl
- Database returns failure if auth=true but no $external db (no sasl)
- Database returns OK:1 if no auth enabled
- If SASL and AUTH:
- SASL checks local cache for entry
- No match, open SSL connection to LDAP for query
- LDAP returns any result
- Store entry in cache
- SASL returns result to Mongo
- Mongo check roles user has to determine if allowed to use resource for commands
- Finally Mongo returns results to driver and application

LDAP in Mongo is easy!

Special **$external** database tells it to check LDAP

- Creating a user is still with .createUser, so nothing changes there, but be sure to use db/collection resource tags

- Auth requires two more fields, however:
  - mechanism: “PLAIN”
  - digestPassword: false

Mongo roles simplified

Four main built-in roles to use

- read
- readWrite
  - Includes read
- dbOwner
  - Includes readWrite, dbAdmin, userAdmin (db level)
- dbAdminAnyDatabase
  - dbAdmin on all DBs but not user admin or readWrite
- root
  - Super user, but with limits*

The missing roles and their issues

- Root is not root
  - Unable to change system collection!
  - Some commands are still not available to this role
  - Need an “any/any” role for true “Super User” (but never let applications use it)
- Wildcard database role to avoid AnyDatabase including “admin”
  - readWriteAnyDatabase is VERY wide and exposes user names and roles to a potential attack via the application user
  - Won’t limit some applications to some databases
  - Prevents multi-tenant with multi databases
  - New database will not automatically be granted access

Making a custom role

- Resources
  - Four types of resources
    - “db”: string or “” for “any” but no wild carding
    - “collection”: string or “” for “any” but no wild carding
    - “cluster”: Boolean True/False
    - “anyResource”: Boolean True/False
- Inherit
  - createRole can have array of “roles” which it will inhert
- Adding new DB’s
  - grandPrivilegesToRole command will add new “resources” to an existing role

Making a custom role: superRoot

```
db = db.getSiblingDB(“admin”);
db.createRole({
      role: “superRoot”,
      privileges:[{
            resource: {anyResource: true},
            actions: ['anyAction']
      }],
      roles: []
});
db.createUser({
      user: “companyDBA”,
      pwd:”EWqeeFpUt9*8zg”,
      roles: [ “superRoot”]
})
```

Making a custom role: multiDB

```
db = db.getSiblingDB(“admin”);
db.createUser({
      user: "app_user",
      pwd : "8JLe!agUg6qEnJ",
      roles: [
            { role: "dbOwner", db: "prod_campaigns" } ,
            { role: "dbOwner", db: "prod_clicks" }
       ]
});
db.runCommand({
      grantRolesToRole: “app_user”,
      roles: [ { role: "dbOwner", db: "prod_publishers" }
      ]
});
```

Mongo and SSL

- Mongo has had SSL support for a long time
  - Legacy with no checking
    - --ssl in config
    - ssl:true in drivers
- Modern
  - Checks for valid hosts (optional)
  - Setup Key file to us
  - Custom CA for self-signed or alternative signers
  - allowSSL, preferSSL, requireSSL modes

SSL: Using a custom CA

If you're just trying to secure the connection, you might be tempted to use “sslAllowInvalidCertficates”. However, this is generally a bad idea for a few reasons:

1) Any failure reason is allowed from expired to revoked certificates
2) You are not ensuring restrictions to a specific hostname
3) You’re not nearly as secure as you think you are

To fix this, simply set net.ssl.CAFile, and Mongo will use BOTH the key and CA file (needs to also be done on the client).

Disk encryption

- Data-at-rest encryption can be solved with any/all of the following:
  - Encrypt the entire volume
  - Encrypt just the database files
  - Encrypt in the application

- Only the first and second options fit most PCI and other certification requirements

- The first item can be solved with disk encryption on the file system, and it very easy to set up using LUKS and dm-encrypt

- The second item is specific to MongoDB Enterprise, but is more than regulations currently need.

Mongo audit trails

- Critical for tracking what has been done on the system for PCI and other certifications
- Percona Server and MongoDB Enterprise both offer this feature and it works the same way
- You simply put in a filter to limit what needs to be logged, then it logs anytime a user or resource is touched
- Now you can use the data to make a timeline in the event of an audit or incident to understand any risk or damage to systems

Reducing your network attack surface

MongoDB's documents suggest you put a mongos on each app host. While this is great for performance reducing one of the network jumps. It actually is very bad for security.

Every mongos must be able to talk to:

- Every primary
- Every secondary
- Every config server

As you can imagine this is a network security nightmare, instead have a group of mongos server near the applications that you let talk to the other systems and the applications can only talk to the mongos. You will have limited the network into a reasonable DMZ type setup.

PERCONA

Thanks for joining!
Be sure to checkout the Percona Blog for more technical blogs and topics!