MongoDB University

MongoDB Security Workshop

MongoDB Security Workshop

Release 3.4

MongoDB, Inc.

May 26, 2017

Contents

1 Security Workshop

Lab: Security Workshop **(page 2)** Securing a full deployment end to end

1.1 Lab: Security Workshop

Note:

- We assume you are familiar with the provisioning scripts. If not start with: https://docs.google.com/document/d/17asAf_1lrj5lUynl6RdyrRekQeGbvsAW5ZV3F3FM3cA
- You will provision 4 nodes per team or attendee, using the provisioning script. For example, for a 9 team or attendee exercise, you would run:

```
deploy.py --profile mdbw-security --teams 9 --instances 4 --noom --run INSTRUCTOR-
↪StetsonD
```

- 3 nodes for the replica set, 1 node for the client application
- Replace INSTRUCTOR with your name, and StetsonD with the room you are presenting in
- The attendees will have to:
- setup SSL with the client certificates
- setup auth
- enable encryption at rest
- enable log redaction
- Each successive objective of this workshop requires full completion of previous objectives. Ensure all participants have met the requirements of a specific section before proceeding.
- TODOs:
- consider "Solution: Enable SSL with the client" being a separate exercise that is done later, as it is more difficult
- consider adding "bonus exercises" for people who finish earlier: things like generating your own certificates

Learning Objectives

Upon completing this workshop, attendees will be able to:

- Secure application communication with MongoDB
- Understand all security authentication and authorization options of MongoDB
- Encrypt MongoDB data at rest using encrypted storage engine
- Feel comfortable deploying and securely configuring MongoDB

Introduction

In this workshop, attendees will install and configure a secure replica set on servers running in AWS.

- We are going to secure the backend communications using TLS/SSL
- Enable authorization on the backend side
- Encrypt the storage layer
- Make sure that there are no *"leaks"* of information

Note: Describe to the students the different components of this workshop.

- The application code in "security-lab/mongo-messenger"
- The 3 AWS instances where the backend should be running
- The single AWS instance where the node app should run
- The set of available files in "/share/downloads":
  - mongodb_packages => MongoDB binaries
  - certs => X509 certificates
  - config => Configuration files
- How to install and start MongoDB on the instances using the package
- Provide the students with the following files:
  - AdvancedAdministrator.pem
  - Info per team about IPs, hostnames, ... from running "describe.py –run security_workshop"
  - certs.tgz

Exercise: Accessing your instances from Windows

- Download and install Putty from **http://www.putty.org/**
- Start Putty with: **All Programs > PuTTY > PuTTY**
- In **Session**:
  - In the **Host Name** box, enter **centos@<publicIP>**
  - Under **Connection type**, select **SSH**
- In **Connection/SSH/Auth**,
  - Browse to the **AdvancedAdministrator.ppk** file
- Click **Open**
- Detailed info at: Connect to AWS with Putty[1]

Note: [TODO] fix agent forwarding.

You can convert .pem files on Mac by:

- brew install putty

[1] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html

- puttygen mykey.pem -o mykey.ppk

Exercise: Accessing your instances from Linux or Mac

- get your .pem file and close the permissions on it

```
chmod 600 AdvancedAdministrator.pem
```

- enable the keychain and `ssh` into `node1`, propagating your credentials

```
ssh-add -K AdvancedAdministrator.pem
ssh -i AdvancedAdministrator.pem -A centos@54.235.1.1
```

- ssh into `node2` from `node1`

```
ssh -A node2
```

Solution: Accessing your instances

In our machines we will have access to all nodes in the deployment:

```
cat /etc/hosts
```

A `/share/downloads` folder with all necessary software downloaded

```
ls /share/downloads
ls /etc/ssl/mongodb
```

Note:

- quickly describe the contents of the dirs under `downloads` Don't emphasize `validation`, this is where your validation scripts are

Exercise: Starting MongoDB and configuring the replica set

- `/share/downloads/mongodb_packages` contains MongoDB 3.2 and 3.4
- installation instructions are at:
  - https://docs.mongodb.com/manual/tutorial/install-mongodb-enterprise-on-red-hat/
- configure the 3 nodes as a replica set named **SECURED**
- use `node1`, `node2` and `node3` for your host names
- you *MUST* use a config file[2]

[2] https://docs.mongodb.com/manual/reference/configuration-options/

Starting MongoDB and configuring the replica set (cont)

- installation

```
sudo yum install -y mongodb-enterprise-server-3.4.2-1.el7.x86_64.rpm
sudo vi /etc/mongod.conf
sudo service mongod start
```

- configure the 3 nodes as a replica set named **SECURED**, change **bindIp** to the **10.0.0.X** address, plus **127.0.0.1**

```
replication:
    replSetName: SECURED
net:
    bindIp: 10.0.0.101,127.0.0.1
```

- initiate the replica set

```
cfg = { _id: "SECURED", version: 1, members: [ {_id: 0, host: "node1:27017"}, {_
↪id: 1, host: "node2:27017"}, {_id:2, host: "node3:27017"} ] }
rs.initiate(cfg)
rs.status()
```

Exercise: Check the Connection to MongoDB

Let's try to connect to our running MongoDB cluster.

```
mongo --host SECURED/node1,node2,node3
```

Note:

- Here we want to make sure everyone can connect correctly to the MongoDB cluster.
- A student may ask why there are no primaries in the replica set. This is most likely because they've reached this step quickly and an election is still taking place

Exercise: Launch the Client Application

It's time to connect our client application.

- install the application:

```
cd ~
tar xzvf /share/downloads/apps/security_lab.tgz
cd mongo-messenger
npm install
npm start
```

- Connect to the public ip of your `node4` instance, port 8080
  - `http://NODE4-public-IP:8080`

Note: The sample application code should be available in the AWS instance. In case students cannot find it, they can download using this instruction:

```
curl -O https://s3.amazonaws.com/mongodb-training/security_lab/security_lab.tgz
```

Also, make sure all instances have `nodejs` installed.

How is the client application connecting to the database?

- the connection string used by the application is in `message.js` and looks like this:

```
const url = "mongodb://node1:27017,node2:27017,node3:27017/
    security-lab?replicaSet=SECURED"
```

- this will work, for now...

WARNING: Spying your deployment!

Throughout the lab, the instructor will be spying on your deployment!

This checking is done by running a few scripts on your machines that will verify whether or not you have completely secured your deployment.

We will come back to this later on.

Note: At this point you should run the lab validation scripts against one of the students environment to test which things are not yet secured.

- run any arbitrary command. Try the following first, as that first run may fail due to the fact that is has to add the hostname to `` `known_hosts ``.

  ```
  ./manage.py --run NathanL-StetsonD --teams all --roles node1,node2,node3 --cmd "/
  ↪bin/hostname -f"
  ./manage.py --run NathanL-StetsonD --teams all --roles node1,node2,node3 --cmd
  ↪"python /share/downloads/validation/validate_log_redaction.py"
  ./manage.py --run NathanL-StetsonD --teams all --roles node1,node2,node3 --cmd
  ↪"sudo python /share/downloads/validation/validate_se_encryption.py"
  ```

Exercise: Set up Authentication

Once we have our sample application up an running is time to start securing the system.

You should start by enabling MongoDB authentication[3]

To do this, you will have to decide:

- Which authentication mechanism to use
- Which authorization support will you use
- Set of users required to operate this system

Note: At this stage we will have the students making decisions regarding the authentication and authorization mechanisms available, and which better suits their needs.

[3] https://docs.mongodb.com/manual/core/authentication/

Ask questions like:

- Which authentication mechanisms should we be setting up?
  - Remind students that we will want the application to be fully encrypted
- Which systems should we have in place to manage users?
  - What happens if we decide to have more than one application in this system?
  - What will happen if we need to remove

Solution: Enable authorization

- using the localhost exception, create the first user

```
use admin
db.createUser({
  user: "foo",
  pwd: "bar",
  roles: [{role: "root", db: "admin"}]
})
db.createUser({
  user: "mongo-messenger",
  pwd: "nodejs",
  roles: [{role: "readWrite", db: "security-lab"}]
})
```

  - must be able to create other users
- need to change “Mongo Messenger” to auth with a given user

Exercise: Enable SSL between the nodes

- we restricted “bindIp” to a local network interface, however if this was an outside address, it would not be good enough
- let’s ensure we limit the connections to a list of nodes we control
  - let’s use SSL certificates
  - they are in `/share/downloads/certs`
- http://mongodb.github.io/node-mongodb-native/2.2/tutorials/connect/ssl/

Solution: Enable SSL between the nodes

- add SSL

```
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: /etc/ssl/mongodb/node1.pem
    CAFile: /etc/ssl/mongodb/ca.pem
security:
  clusterAuthMode: x509
```

- restart all running mongod's
- connect with the client
- why no `authorization: enabled` in the config file?
  - Enabling clusterAuth implicity enables authorization

```
mongo --ssl --host node1 --sslCAFile /etc/ssl/mongodb/ca.pem --sslPEMKeyFile /etc/ssl/
↪mongodb/node1.pem
```

Solution: Enable SSL with the client

- In `~/mongo-messenger/messages.js`

```
const fs = require('fs');

// Read the certificates and create ssl options object
let ca = [fs.readFileSync('/etc/ssl/mongodb/ca.pem')];
let cert = fs.readFileSync('/etc/ssl/mongodb/node1.pem');
let key = fs.readFileSync('/etc/ssl/mongodb/node1.pem');
let options = {
  sslValidate:true,
  sslCA:ca,
  sslCert:cert,
  sslKey:key
}
// mongo-messenger user
let user = encodeURIComponent('mongo-messenger');
let pass = encodeURIComponent('nodejs');
const url = `mongodb://${user}:${pass}@node1:27017,node2:27017,node3:27017/security-
↪lab?authSource=admin&replicaSet=SECURED&ssl=true`

let messages = null;

MongoClient.connect(url, options, (err, db) => {
  assert.equal(null, err)
  messages = db.collection("messages")
})
```

Troubleshooting

- Student receives an error that the primary can't be found
  - Their connection url is most likely malformed.
- Student receives an error that they don't have permission on the collection
  - If they've correctly set up authorization and role-based access control, they need to add the application user credentials and specify an `authSource`

Exercise: Encrypt Storage Layer

To fully secure our MongoDB deployment we need to consider the actual MongoDB instance files. Your instructor has some scripts that will enable him to have a peek into the your collection and indexes data files.

Don't let him do so!!!

Solution: Encrypt Storage Layer

- enable encryption on the storage engine

```
security:
  enableEncryption: true
  encryptionKeyFile: /etc/ssl/mongodb/mongodb-keyfile
```

- different options to manage encryption keys, however for this lab a simple keyfile should have been enough

```
openssl rand -base64 32 > mongodb-keyfile
chmod 600 mongodb-keyfile
```

https://docs.mongodb.com/manual/tutorial/configure-encryption/

Exercise: Avoid any log leaks

Logs are an important asset of your system.

Allow us to understand any potential issue with our cluster or deployment. But they can also **leak** some confidential information!

Make sure that you do not have any data leaks into your logs.

This should be done without downtime

Note: At this point students should enable log redaction in their cluster nodes.

To accomplish this students should do the following:

- relaunch all nodes enabling client log data redaction
- Give "extra kudos" to students that managed to do this and also clearing any information on previous logs

Solution: Avoid any log leaks

by setting up log redaction

```
security:
  redactClientLogData: true
```

```
db.adminCommand(
  { setParameter: 1, redactClientLogData : true }
)
```

- and you need a way to prove/show that redaction worked

Auditing

At this point we have a secured MongoDB deployment hardened against outside attacks, and used Role-Based Access Control to limit the access of users. The final step is to enable auditing, giving us a clear record of **who** performed an auditable action.

Exercise: Enable Auditing

- Enable auditing for all operations, to include CRUD operations, for your mongo-messenger user
- Output the log file in JSON format
- A log file has been created for you at `/mongod-data/audit/SECURED/audit.json`
- There are many filter options[4]

Note:

- Outputting to a BSON file will preserve the most information with the best performance, however we are outputting in JSON so participants can more easily parse the audit log.
- Take care when auditing CRUD operations, they are not redacted in the audit log
- Talk to the students about the different audit formats and destinations, touching on the pros and cons (e.g. readability, log trucation, and performance implications)
  - JSON, BSON
  - File, syslog, console
- Logging successful authentication actions is slower than logging just failed authentication actions
- Auditable actions can be fine-tuned in the audit filter parameter

---

[4] https://docs.mongodb.com/manual/tutorial/configure-audit-filters/

Solution: Enable Auditing

```
setParameter: { auditAuthorizationSuccess: true }

auditLog:
  destination: "file"
  format: "JSON"
  path: /var/log/SECURED/audit.json
  filter: '{ users: { user: "mongo-messenger", db: "security-lab" } }'
```

Putting it together

```
net:
  ssl:
    mode: requireSSL
    PEMKeyFile: /etc/ssl/mongodb/node1.pem
    CAFile: /etc/ssl/mongodb/ca.pem

security:
  clusterAuthMode: x509
  enableEncryption : true
  encryptionKeyFile : /etc/ssl/mongodb/mongodb-keyfile
  redactClientLogData: true

setParameter: { auditAuthorizationSuccess: true }

auditLog:
  destination: "file"
  format: "JSON"
  path: /mongod-data/audit/SECURED/audit.json
  filter: '{ users: { user: "mongo-messenger", db: "security-lab" } }'
```

Summary

What we did:

- enabled basic authorization
- used SSL certificates
- encrypted the database at rest
- redacted the mongod logs
- configured auditing for a specific user

Note: Ask what else could be done?

- MongoDB
  - link authorization and authentication to outside system like LDAP or Kerberos
  - enable redaction of results
- Infrastructure
  - use security groups if in AWS

    - use firewall
- Application
    - security at the application level