MongoDB Security Guide

Release 3.0.6

MongoDB Documentation Project

September 08, 2015

Contents

This section outlines basic security and risk management strategies and access control. The included tutorials outline specific tasks for configuring firewalls, authentication, and system privileges.

Security Introduction **(page 3)** A high-level introduction to security and MongoDB deployments.

Network Security **(page 5)** Documentation on authentication, authorization, and encryption in MongoDB.

Access Control **(page 14)** Documentation on users and roles in MongoDB.

Auditing **(page 16)** Documentation on the auditing feature available with MongoDB Enterprise.

External Environment **(page 17)** Discusses potential risks related to MongoDB's JavaScript, HTTP and REST interfaces, including strategies to control those risks.

Security Tutorials **(page 18)** Tutorials for enabling and configuring security features for MongoDB.

Network Security Tutorials **(page 19)** Ensure that the underlying network configuration supports a secure operating environment for MongoDB deployments, and appropriately limits access to MongoDB deployments.

Authentication Tutorials **(page 39)** These tutorials describe procedures relevant for the configuration, operation, and maintenance of MongoDB's access control system.

User and Role Management Tutorials **(page 66)** MongoDB's access control system provides a flexible role-based access control system that you can use to limit access to MongoDB deployments. The tutorials in this section describe the configuration an setup of the authorization system.

Continue reading from *Security Tutorials* (page 18) for additional tutorials that address the use and management of secure MongoDB deployments.

Create a Vulnerability Report **(page 83)** Report a vulnerability in MongoDB.

Security Reference **(page 84)** Reference for security related functions.

Security Checklist **(page 111)** A high level overview of global security consideration for administrators of MongoDB deployments. Use this checklist if you are new to deploying MongoDB in production and want to implement high quality security practices.

1 Security Introduction

Maintaining a secure MongoDB deployment requires administrators to implement controls to ensure that users and applications have access to only the data that they require. MongoDB provides features that allow administrators to implement these controls and restrictions for any MongoDB deployment.

If you are already familiar with security and MongoDB security practices, consider the *Security Checklist* (page 111) for a collection of recommended actions to protect a MongoDB deployment.

1.1 Authentication

Before gaining access to a system all clients should identify themselves to MongoDB. This ensures that no client can access the data stored in MongoDB without being explicitly allowed.

MongoDB supports a number of *authentication mechanisms* (page 6) that clients can use to verify their identity. MongoDB supports two mechanisms: a password-based challenge and response protocol and x.509 certificates. Additionally, MongoDB Enterprise[1] also provides support for *LDAP proxy authentication* (page 8) and *Kerberos authentication* (page 8).

[1] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs

See *Authentication* (page 6) for more information.

1.2 Role Based Access Control

Access control, i.e. *authorization* (page 14), determines a user's access to resources and operations. Clients should only be able to perform the operations required to fulfill their approved functions. This is the "principle of least privilege" and limits the potential risk of a compromised application.

MongoDB's role-based access control system allows administrators to control all access and ensure that all granted access applies as narrowly as possible. MongoDB does not enable authorization by default. When you enable *authorization* (page 14), MongoDB will require authentication for all connections.

When authorization is enabled, MongoDB controls a user's access through the roles assigned to the user. A role consists of a set of privileges, where a privilege consists of *actions*, or a set of operations, and a *resource* upon which the actions are allowed.

Users may have one or more role that describes their access. MongoDB provides several *built-in roles* (page 85) and users can construct specific roles tailored to clients' actual requirements.

See *Authorization* (page 14) for more information.

1.3 Auditing

Auditing provides administrators with the ability to verify that the implemented security policies are controlling activity in the system. Retaining audit information ensures that administrators have enough information to perform forensic investigations and comply with regulations and polices that require audit data.

See *Auditing* (page 16) for more information.

1.4 Encryption

Transport Encryption

You can use TLS/SSL (Transport Layer Security/Secure Sockets Layer) to encrypt all of MongoDB's network traffic. TLS/SSL ensures that MongoDB network traffic is only readable by the intended client.

See *Configure mongod and mongos for TLS/SSL* (page 26) for more information.

Encryption at Rest

There are two broad classes of approaches to encrypting data at rest with MongoDB: Application Level Encryption (page 4) and Storage Encryption (page 4). You can use these solutions together or independently. **Application Level Encryption** provides encryption on a per-field or per-document basis within the application layer. To encrypt document or field level data, write custom encryption and decryption routines or use a commercial solution such as the Vormetric Data Security Platform[2]. **Storage Encryption** encrypts all MongoDB data on the storage or operating system to ensure that only authorized processes can access protected data. A number of third-party libraries can integrate with the operating system to provide transparent disk-level encryption. For example:

- **Linux Unified Key Setup (LUKS)** LUKS is available for most Linux distributions. For configuration explanation, see the LUKS documentation from Red Hat[3].
- **IBM Guardium Data Encryption** IBM Guardium Data Encryption[4] provides support for disk-level encryp-

[2] http://www.vormetric.com/sites/default/files/sb-MongoDB-Letter-2014-0611.pdf
[3] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Security_Guide/sec-Encryption.html
[4] http://www-03.ibm.com/software/products/en/infosphere-guardium-data-encryption

tion for Linux and Windows operating systems.

- **Vormetric Data Security Platform** The Vormetric Data Security Platform[5] provides disk and file-level encryption in addition to application level encryption.
- **Bitlocker Drive Encryption** Bitlocker Drive Encryption[6] is a feature available on Windows Server 2008 and 2012 that provides disk encryption.

Properly configured disk encryption, when used alongside good security policies that protect relevant accounts, passwords, and encryption keys, can help ensure compliance with standards, including HIPAA, PCI-DSS, and FERPA.

1.5 Hardening Deployments and Environments

In addition to implementing controls within MongoDB, you should also place controls around MongoDB to reduce the risk exposure of the entire MongoDB system. This is a *defense in depth* strategy.

Hardening MongoDB extends the ideas of least privilege, auditing, and encryption outside of MongoDB. Reducing risk includes: configuring the network rules to ensure that only trusted hosts have access to MongoDB, and that the MongoDB processes only have access to the parts of the filesystem required for operation.

1.6 Additional Resources

- Making HIPAA Compliant MongoDB Applications[7]
- Security Architecture White Paper[8]
- Webinar: Securing Your MongoDB Deployment[9]

2 Security Concepts

These documents introduce and address concepts and strategies related to security practices in MongoDB deployments.

Network Security **(page 5)** Documentation on authentication, authorization, and encryption in MongoDB.

Access Control **(page 14)** Documentation on users and roles in MongoDB.

Auditing **(page 16)** Documentation on the auditing feature available with MongoDB Enterprise.

External Environment **(page 17)** Discusses potential risks related to MongoDB's JavaScript, HTTP and REST interfaces, including strategies to control those risks.

2.1 Network Security

These documents introduce and address concepts and strategies related to authentication, authorization, and encryption.

Authentication **(page 6)** Mechanisms for verifying user and instance access to MongoDB.

Network Exposure and Security **(page 9)** Discusses potential security risks related to the network and strategies for decreasing possible network-based attack vectors for MongoDB.

[5] http://www.vormetric.com/sites/default/files/sb-MongoDB-Letter-2014-0611.pdf
[6] http://technet.microsoft.com/en-us/library/hh831713.aspx
[7] https://www.mongodb.com/blog/post/making-hipaa-compliant-applications-mongodb?jmp=docs
[8] https://www.mongodb.com/lp/white-paper/mongodb-security-architecture?jmp=docs
[9] http://www.mongodb.com/presentations/webinar-securing-your-mongodb-deployment?jmp=docs

Kerberos Authentication **(page 11)** Kerberos authentication and MongoDB.

Authentication

Authentication is the process of verifying the identity of a client. When access control, i.e. *authorization* (page 14), is enabled, MongoDB requires all clients to authenticate themselves first in order to determine the access for the client.

Although authentication and *authorization* (page 14) are closely connected, authentication is distinct from authorization. Authentication verifies the identity of a user; authorization determines the verified user's access to resources and operations.

MongoDB supports a number of *authentication mechanisms* (page 6) that clients can use to verify their identity. These mechanisms allow MongoDB to integrate into your existing authentication system. See *Authentication Mechanisms* (page 6) for details.

In addition to verifying the identity of a client, MongoDB can require members of replica sets and sharded clusters to *authenticate their membership* (page 8) to their respective replica set or sharded cluster. See *Authentication Between MongoDB Instances* (page 8) for more information.

Client Users

To authenticate a client in MongoDB, you must add a corresponding user to MongoDB. When adding a user, you create the user in a specific database. This database is the *authentication database* for the user.

Together, the user's name and database serve as a unique identifier for that user. That is, if two users have the same name but are created in different databases, they are two separate users. To authenticate, the client must authenticate the user against the user's *authentication database*. For instance, if using the `mongo` shell as a client, you can specify the authentication database for the user with the *–authenticationDatabase* option.

To add and manage user information, MongoDB provides the `db.createUser()` method as well as other *user management methods*. For examples of user management in MongoDB, see *Manage User and Roles* (page 69).

MongoDB stores all user information, including `name` (page 96), `password` (page 96), and the `user's database` (page 96), in the *system.users* (page 96) collection in the `admin` database.

Authentication Mechanisms

Changed in version 3.0.

MongoDB supports multiple authentication mechanisms. MongoDB's default authentication method is a *challenge and response mechanism (SCRAM-SHA-1)* (page 6). Previously, MongoDB used *MongoDB Challenge and Response (MONGODB-CR)* (page 7) as the default.

MongoDB also supports *x509 certificate authentication* (page 8), *LDAP proxy authentication* (page 8), and *Kerberos authentication* (page 8).

This section introduces the mechanisms available in MongoDB.

To specify the authentication mechanism to use, see `authenticationMechanisms`.

`SCRAM-SHA-1` Authentication New in version 3.0.

`SCRAM-SHA-1` is an IETF standard, RFC 5802[10], that defines best practice methods for implementation of challenge-response mechanisms for authenticating users with passwords.

[10] https://tools.ietf.org/html/rfc5802

SCRAM-SHA-1 verifies supplied user credentials against the user's name (page 96), password (page 96) and database (page 96). The user's database is the database where the user was created, and the user's database and the user's name together serves to identify the user.

Note: A driver upgrade is **necessary** to use the SCRAM-SHA-1 authentication mechanism if your current driver version does not support SCRAM-SHA-1. See *required driver versions* for details.

See also:

- Blog Post: Improved Password-Based Authentication in MongoDB 3.0: SCRAM Explained (Part 1)[11]
- Blog Post: Improved Password-Based Authentication in MongoDB 3.0: SCRAM Explained (Part 2)[12]

SCRAM-SHA-1 Advantages MongoDB's implementation of SCRAM-SHA-1 represents an improvement in security over the previously-used MONGODB-CR, providing:

- A tunable work factor (iterationCount),
- Per-user random salts rather than server-wide salts,
- A cryptographically stronger hash function (SHA-1 rather than MD5), and
- Authentication of the server to the client as well as the client to the server.

SCRAM-SHA-1 and Existing User Credentials SCRAM-SHA-1 is the default mechanism for MongoDB versions beginning with the 3.0 series. However, if you are upgrading a MongoDB 2.6 instances that already have users credentials, MongoDB will continue to use MONGODB-CR for challenge-response authentication until you upgrade the authentication schema. Even when using the MONGODB-CR authentication mechanism, clients and drivers that support MongoDB 3.0 features (see *compatibility-driver-versions*) will use the SCRAM communication protocol.

For details on upgrading the authentication schema model to SCRAM-SHA-1, see http://docs.mongodb.org/manual/release-notes/3.0-scram.

Warning: The procedure to upgrade to SCRAM-SHA-1 **discards** the MONGODB-CR credentials used by 2.6. As such, the procedure is **irreversible**, short of restoring from backups.
The procedure also disables MONGODB-CR as an authentication mechanism.

MONGODB-CR Authentication MONGODB-CR is a challenge-response mechanism that authenticates users through passwords.

Changed in version 3.0: As of version 3.0, MongoDB no longer defaults to MONGODB-CR and instead uses SCRAM-SHA-1 as the default authentication mechanism.

MONGODB-CR verifies supplied user credentials against the user's name (page 96), password (page 96) and database (page 96). The user's database is the database where the user was created, and the user's database and the user's name together serve to identify the user.

Using key files, you can also use MONGODB-CR authentication for the *internal member authentication* (page 8) of replica set members and sharded cluster members. The contents of the key files serve as the shared password for the members. You must store the key file on each mongod or mongos instance for that replica set or sharded cluster. The content of the key file is arbitrary but must be the same on all mongod and mongos instances that connect to each other.

See *Generate a Key File* (page 62) for instructions on generating a key file and turning on key file authentication for members.

[11] https://www.mongodb.com/blog/post/improved-password-based-authentication-mongodb-30-scram-explained-part-1?jmp=docs
[12] https://www.mongodb.com/blog/post/improved-password-based-authentication-mongodb-30-scram-explained-part-2?jmp=docs

x.509 Certificate Authentication New in version 2.6.

MongoDB supports x.509 certificate authentication for use with a secure *TLS/SSL connection* (page 26).

To authenticate to servers, clients can use x.509 certificates instead of usernames and passwords. See *Client x.509 Certificate* (page 44) for more information.

For membership authentication, members of sharded clusters and replica sets can use x.509 certificates instead of key files. See *Use x.509 Certificate for Membership Authentication* (page 46) for more information.

Kerberos Authentication MongoDB Enterprise[13] supports authentication using a Kerberos service. Kerberos is an industry standard authentication protocol for large client/server systems.

To use MongoDB with Kerberos, you must have a properly configured Kerberos deployment, configured *Kerberos service principals* (page 12) for MongoDB, and added *Kerberos user principal* (page 12) to MongoDB.

See *Kerberos Authentication* (page 11) for more information on Kerberos and MongoDB. To configure MongoDB to use Kerberos authentication, see *Configure MongoDB with Kerberos Authentication on Linux* (page 55) and *Configure MongoDB with Kerberos Authentication on Windows* (page 59).

LDAP Proxy Authority Authentication MongoDB Enterprise[14] supports proxy authentication through a Lightweight Directory Access Protocol (LDAP) service. See *Authenticate Using SASL and LDAP with OpenLDAP* (page 52) and *Authenticate Using SASL and LDAP with ActiveDirectory* (page 49).

MongoDB Enterprise for Windows does **not** include LDAP support for authentication. However, MongoDB Enterprise for Linux supports using LDAP authentication with an ActiveDirectory server.

MongoDB does **not** support LDAP authentication in mixed sharded cluster deployments that contain both version 2.4 and version 2.6 shards.

Authentication Behavior

Client Authentication Clients can authenticate using the *challenge and response* (page 7), *x.509* (page 8), *LDAP Proxy* (page 8) and *Kerberos* (page 8) mechanisms.

Each client connection should authenticate as exactly one user. If a client authenticates to a database as one user and later authenticates to the same database as a different user, the second authentication invalidates the first. While clients can authenticate as multiple users if the users are defined on different databases, we recommend authenticating as one user at a time, providing the user with appropriate privileges on the databases required by the user.

See *Authenticate to a MongoDB Instance or Cluster* (page 61) for more information.

Authentication Between MongoDB Instances You can authenticate members of *replica sets* and *sharded clusters*. To authenticate members of a single MongoDB deployment to each other, MongoDB can use the `keyFile` and *x.509* (page 8) mechanisms. Using `keyFile` authentication for members also enables authorization.

Always run replica sets and sharded clusters in a trusted networking environment. Ensure that the network permits only trusted traffic to reach each `mongod` and `mongos` instance.

Use your environment's firewall and network routing to ensure that traffic *only* from clients and other members can reach your `mongod` and `mongos` instances. If needed, use virtual private networks (VPNs) to ensure secure connections over wide area networks (WANs).

Always ensure that:

[13] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs
[14] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs

- Your network configuration will allow every member of the replica set or sharded cluster to contact every other member.
- If you use MongoDB's authentication system to limit access to your infrastructure, ensure that you configure a `keyFile` on all members to permit authentication.

See *Generate a Key File* (page 62) for instructions on generating a key file and turning on key file authentication for members. For an example of using key files for sharded cluster authentication, see *Enable Authentication in a Sharded Cluster* (page 41).

Authentication on Sharded Clusters In sharded clusters, applications authenticate to directly to `mongos` instances, using credentials stored in the `admin` database of the *config servers*. The shards in the sharded cluster also have credentials, and clients can authenticate directly to the shards to perform maintenance directly on the shards. In general, applications and clients should connect to the sharded cluster through the `mongos`.

Changed in version 2.6: Previously, the credentials for authenticating to a database on a cluster resided on the *primary shard* for that database.

Some maintenance operations, such as `cleanupOrphaned`, `compact`, `rs.reconfig()`, require direct connections to specific shards in a sharded cluster. To perform these operations with authentication enabled, you must connect directly to the shard and authenticate as a *shard local* administrative user. To create a *shard local* administrative user, connect directly to the shard and create the user. MongoDB stores *shard local* users in the `admin` database of the shard itself. These *shard local* users are completely independent from the users added to the sharded cluster via `mongos`. *Shard local* users are local to the shard and are inaccessible by `mongos`. Direct connections to a shard should only be for shard-specific maintenance and configuration.

Localhost Exception The localhost exception allows you to enable authorization *before* creating the first user in the system. When active, the localhost exception allows connections from the localhost interface to create the first user on the `admin` database. The exception applies only when there are no users created in the MongoDB instance.

Changed in version 3.0: The localhost exception changed so that these connections *only* have access to create the first user on the `admin` database. In previous versions, connections that gained access using the localhost exception had unrestricted access to the MongoDB instance.

If you use the localhost exception when deploying a new MongoDB system, the first user you create must be in the `admin` database with privileges to create other users, such as a user with the `userAdmin` (page 87) or `userAdminAnyDatabase` (page 92) role. See *Enable Client Access Control* (page 40) and *Create a User Administrator* (page 67) for more information.

In the case of a sharded cluster, the localhost exception applies to each shard individually as well as to the cluster as a whole. Once you create a sharded cluster and add an administrator to the `mongos` instance, you must still prevent unauthorized access to the individual shards. Follow one of the following steps for each shard in your cluster:

- Create an administrative user, or
- Disable the localhost exception at startup. To disable the localhost exception, use `setParameter` in your `configuration file`, or `--setParameter` on the command line to set the `enableLocalhostAuthBypass` parameter to `0`.

Network Exposure and Security

By default, MongoDB programs (i.e. `mongos` and `mongod`) will bind to all available network interfaces (i.e. IP addresses) on a system.

This page outlines various runtime options that allow you to limit access to MongoDB programs.

Configuration Options

You can limit the network exposure with the following `mongod` and `mongos` configuration options: `enabled`, `net.http.RESTInterfaceEnabled`, `bindIp`, and `port`. You can use a `configuration file` to specify these settings.

`nohttpinterface` The `enabled` setting for `mongod` and `mongos` instances disables the "home" status page.

Changed in version 2.6: The `mongod` and `mongos` instances run with the http interface *disabled* by default.

The status interface is read-only by default, and the default port for the status page is `28017`. Authentication does not control or affect access to this interface.

Warning: Disable this interface for production deployments. If you *enable* this interface, you should only allow trusted clients to access this port. See *Firewalls* (page 11).

`rest` The `net.http.RESTInterfaceEnabled` setting for `mongod` enables a fully interactive administrative *REST* interface, which is *disabled* by default. The `net.http.RESTInterfaceEnabled` configuration makes the http status interface [15], which is read-only by default, fully interactive. Use the `net.http.RESTInterfaceEnabled` setting with the `enabled` setting.

The REST interface does not support any authentication and you should always restrict access to this interface to only allow trusted clients to connect to this port.

You may also enable this interface on the command line as *`mongod --rest --httpinterface`*.

Warning: Disable this option for production deployments. If *do* you leave this interface enabled, you should only allow trusted clients to access this port.

`bind_ip` The `bindIp` setting for `mongod` and `mongos` instances limits the network interfaces on which MongoDB programs will listen for incoming connections. You can also specify a number of interfaces by passing `bindIp` a comma separated list of IP addresses. You can use the *`mongod --bind_ip`* and *`mongos --bind_ip`* option on the command line at run time to limit the network accessibility of a MongoDB program.

Important: Make sure that your `mongod` and `mongos` instances are only accessible on trusted networks. If your system has more than one network interface, bind MongoDB programs to the private or internal network interface.

`port` The `port` setting for `mongod` and `mongos` instances changes the main port on which the `mongod` or `mongos` instance listens for connections. The default port is `27017`. Changing the port does not meaningfully reduce risk or limit exposure. You may also specify this option on the command line as *`mongod --port`* or *`mongos --port`*. Setting `port` also indirectly sets the port for the HTTP status interface, which is always available on the port numbered `1000` greater than the primary `mongod` port.

Only allow trusted clients to connect to the port for the `mongod` and `mongos` instances. See *Firewalls* (page 11).

See also *configuration-security* and *Default MongoDB Port* (page 104).

[15] Starting in version 2.6, http interface is *disabled* by default.

Firewalls

Firewalls allow administrators to filter and control access to a system by providing granular control over what network communications. For administrators of MongoDB, the following capabilities are important: limiting incoming traffic on a specific port to specific systems, and limiting incoming traffic from untrusted hosts.

On Linux systems, the `iptables` interface provides access to the underlying `netfilter` firewall. On Windows systems, `netsh` command line interface provides access to the underlying Windows Firewall. For additional information about firewall configuration, see *Configure Linux iptables Firewall for MongoDB* (page 20) and *Configure Windows netsh Firewall for MongoDB* (page 23).

For best results and to minimize overall exposure, ensure that *only* traffic from trusted sources can reach `mongod` and `mongos` instances and that the `mongod` and `mongos` instances can only connect to trusted outputs.

See also:

For MongoDB deployments on Amazon's web services, see the Amazon EC2[16] page, which addresses Amazon's Security Groups and other EC2-specific security features.

Virtual Private Networks

Virtual private networks, or VPNs, make it possible to link two networks over an encrypted and limited-access trusted network. Typically, MongoDB users who use VPNs use TLS/SSL rather than IPSEC VPNs for performance issues.

Depending on configuration and implementation, VPNs provide for certificate validation and a choice of encryption protocols, which requires a rigorous level of authentication and identification of all clients. Furthermore, because VPNs provide a secure tunnel, by using a VPN connection to control access to your MongoDB instance, you can prevent tampering and "man-in-the-middle" attacks.

Kerberos Authentication

New in version 2.4.

Overview

MongoDB Enterprise provides support for Kerberos authentication of MongoDB clients to `mongod` and `mongos`. Kerberos is an industry standard authentication protocol for large client/server systems. Kerberos allows MongoDB and applications to take advantage of existing authentication infrastructure and processes.

Kerberos Components and MongoDB

Principals In a Kerberos-based system, every participant in the authenticated communication is known as a "principal", and every principal must have a unique name.

Principals belong to administrative units called *realms*. For each realm, the Kerberos Key Distribution Center (KDC) maintains a database of the realm's principal and the principals' associated "secret keys".

For a client-server authentication, the client requests from the KDC a "ticket" for access to a specific asset. KDC uses the client's secret and the server's secret to construct the ticket which allows the client and server to mutually authenticate each other, while keeping the secrets hidden.

For the configuration of MongoDB for Kerberos support, two kinds of principal names are of interest: *user principals* (page 12) and *service principals* (page 12).

[16] https://docs.mongodb.org/ecosystem/platforms/amazon-ec2

User Principal To authenticate using Kerberos, you must add the Kerberos user principals to MongoDB to the `$external` database. User principal names have the form:

```
<username>@<KERBEROS REALM>
```

For every user you want to authenticate using Kerberos, you must create a corresponding user in MongoDB in the `$external` database.

For examples of adding a user to MongoDB as well as authenticating as that user, see *Configure MongoDB with Kerberos Authentication on Linux* (page 55) and *Configure MongoDB with Kerberos Authentication on Windows* (page 59).

See also:

User and Role Management Tutorials (page 66) for general information regarding creating and managing users in MongoDB.

Service Principal Every MongoDB `mongod` and `mongos` instance (or `mongod.exe` or `mongos.exe` on Windows) must have an associated service principal. Service principal names have the form:

```
<service>/<fully qualified domain name>@<KERBEROS REALM>
```

For MongoDB, the `<service>` defaults to `mongodb`. For example, if `m1.example.com` is a MongoDB server, and `example.com` maintains the `EXAMPLE.COM` Kerberos realm, then `m1` should have the service principal name `mongodb/m1.example.com@EXAMPLE.COM`.

To specify a different value for `<service>`, use `serviceName` during the start up of `mongod` or `mongos` (or `mongod.exe` or `mongos.exe`). `mongo` shell or other clients may also specify a different service principal name using `serviceName`.

Service principal names must be reachable over the network using the fully qualified domain name (FQDN) part of its service principal name.

By default, Kerberos attempts to identify hosts using the `/etc/krb5.conf` file before using DNS to resolve hosts.

On Windows, if running MongoDB as a service, see *Assign Service Principal Name to MongoDB Windows Service* (page 61).

Linux Keytab Files Linux systems can store Kerberos authentication keys for a *service principal* (page 12) in *keytab* files. Each Kerberized `mongod` and `mongos` instance running on Linux must have access to a keytab file containing keys for its *service principal* (page 12).

To keep keytab files secure, use file permissions that restrict access to only the user that runs the `mongod` or `mongos` process.

Tickets On Linux, MongoDB clients can use Kerberos's `kinit` program to initialize a credential cache for authenticating the user principal to servers.

Windows Active Directory Unlike on Linux systems, `mongod` and `mongos` instances running on Windows do not require access to keytab files. Instead, the `mongod` and `mongos` instances read their server credentials from a credential store specific to the operating system.

However, from the Windows Active Directory, you can export a keytab file for use on Linux systems. See Ktpass[17] for more information.

[17] http://technet.microsoft.com/en-us/library/cc753771.aspx

Authenticate With Kerberos To configure MongoDB for Kerberos support and authenticate, see *Configure MongoDB with Kerberos Authentication on Linux* (page 55) and *Configure MongoDB with Kerberos Authentication on Windows* (page 59).

Operational Considerations

The HTTP Console The MongoDB HTTP Console[18] interface does not support Kerberos authentication.

DNS Each host that runs a `mongod` or `mongos` instance must have both `A` and `PTR` DNS records to provide forward and reverse lookup.

Without `A` and `PTR` DNS records, the host cannot resolve the components of the Kerberos domain or the Key Distribution Center (KDC).

System Time Synchronization To successfully authenticate, the system time for each `mongod` and `mongos` instance must be within 5 minutes of the system time of the other hosts in the Kerberos infrastructure.

Kerberized MongoDB Environments

Driver Support The following MongoDB drivers support Kerberos authentication:

- C[19]
- C++[20]
- Java[21]
- C#[22]
- Node.js[23]
- PHP[24]
- Python[25]
- Ruby[26]

Use with Additional MongoDB Authentication Mechanism Although MongoDB supports the use of Kerberos authentication with other authentication mechanisms, only add the other mechanisms as necessary. See the `Incorporate Additional Authentication Mechanisms` section in *Configure MongoDB with Kerberos Authentication on Linux* (page 55) and *Configure MongoDB with Kerberos Authentication on Windows* (page 59) for details.

[18] https://docs.mongodb.org/ecosystem/tools/http-interfaces/#http-console
[19] https://api.mongodb.org/c/current/authentication.html#kerberos
[20] https://docs.mongodb.org/ecosystem/tutorial/authenticate-with-cpp-driver/
[21] https://docs.mongodb.org/ecosystem/tutorial/authenticate-with-java-driver/
[22] http://mongodb.github.io/mongo-csharp-driver/2.0/reference/driver/authentication/#gssapi-kerberos
[23] http://mongodb.github.io/node-mongodb-native/2.0/tutorials/enterprise_features/
[24] http://php.net/manual/en/mongoclient.construct.php
[25] http://api.mongodb.org/python/current/examples/authentication.html
[26] https://docs.mongodb.org/ecosystem/tutorial/ruby-driver-tutorial/#gssapi-kerberos-mechanism

Additional Resources

- MongoDB LDAP and Kerberos Authentication with Dell (Quest) Authentication Services[27]
- MongoDB with Red Hat Enterprise Linux Identity Management and Kerberos[28]

2.2 Access Control

These documents introduce and address concepts and strategies related to Role Based Access Control in MongoDB.

Authorization **(page 14)** Introduction to Role Based Access Control used in MongoDB

Collection-Level Access Control **(page 16)** Specify collection-level access control.

Authorization

MongoDB employs Role-Based Access Control (RBAC) to govern access to a MongoDB system. A user is granted one or more *roles* (page 14) that determine the user's access to database resources and operations. Outside of role assignments, the user has no access to the system.

MongoDB does not enable authorization by default. You can enable authorization using the `--auth` or the `--keyFile` options, or if using a configuration file, with the `security.authorization` or the `security.keyFile` settings.

MongoDB provides *built-in roles* (page 85), each with a dedicated purpose for a common use case. Examples include the `read` (page 85), `readWrite` (page 86), `dbAdmin` (page 86), and `root` (page 93) roles.

Administrators also can create new roles and privileges to cater to operational needs. Administrators can assign privileges scoped as granularly as the collection level.

When granted a role, a user receives all the privileges of that role. A user can have several roles concurrently, in which case the user receives the union of all the privileges of the respective roles.

Roles

A role consists of privileges that pair resources with allowed operations. Each privilege is specified explicitly in the role or inherited from another role or both.

Except for roles created in the `admin` database, a role can only include privileges that apply to its database and can only inherit from other roles in its database.

A role created in the `admin` database can include privileges that apply to the `admin` database, other databases or to the *cluster* (page 98) resource, and can inherit from roles in other databases as well as the `admin` database.

A user assigned a role receives all the privileges of that role. The user can have multiple roles and can have different roles on different databases.

Roles always grant privileges and never limit access. For example, if a user has both `read` (page 85) *and* `readWriteAnyDatabase` (page 92) roles on a database, the greater access prevails.

[27] https://www.mongodb.com/blog/post/mongodb-ldap-and-kerberos-authentication-dell-quest-authentication-services?jmp=docs
[28] http://docs.mongodb.org/ecosystem/tutorial/manage-red-hat-enterprise-linux-identity-management?jmp=docs

Privileges A privilege consists of a specified resource and the actions permitted on the resource.

A privilege *resource* (page 97) is either a database, collection, set of collections, or the cluster. If the cluster, the affiliated actions affect the state of the system rather than a specific database or collection.

An *action* (page 99) is a command or method the user is allowed to perform on the resource. A resource can have multiple allowed actions. For available actions see *Privilege Actions* (page 99).

For example, a privilege that includes the `update` (page 99) action allows a user to modify existing documents on the resource. To additionally grant the user permission to create documents on the resource, the administrator would add the `insert` (page 99) action to the privilege.

For privilege syntax, see `admin.system.roles.privileges` (page 94).

Inherited Privileges A role can include one or more existing roles in its definition, in which case the role inherits all the privileges of the included roles.

A role can inherit privileges from other roles in its database. A role created on the `admin` database can inherit privileges from roles in any database.

User-Defined Roles New in version 2.6.

User administrators can create custom roles to ensure collection-level and command-level granularity and to adhere to the policy of *least privilege*. Administrators create and edit roles using the *role management commands*.

MongoDB scopes a user-defined role to the database in which it is created and uniquely identifies the role by the pairing of its name and its database. MongoDB stores the roles in the `admin` database's *system.roles* (page 93) collection. Do not access this collection directly but instead use the *role management commands* to view and edit custom roles.

Collection-Level Access Control By creating a role with *privileges* (page 15) that are scoped to a specific collection in a particular database, administrators can implement collection-level access control.

See *Collection-Level Access Control* (page 16) for more information.

Users

MongoDB stores user credentials in the protected `admin.system.users`. Use the *user management methods* to view and edit user credentials.

Role Assignment to Users User administrators create the users that access the system's databases. MongoDB's *user management commands* let administrators create users and assign them roles.

MongoDB scopes a user to the database in which the user is created. MongoDB stores all user definitions in the `admin` database, no matter which database the user is scoped to. MongoDB stores users in the `admin` database's *system.users collection* (page 96). Do not access this collection directly but instead use the *user management commands*.

The first role assigned in a database should be either `userAdmin` (page 87) or `userAdminAnyDatabase` (page 92). This user can then create all other users in the system. See *Create a User Administrator* (page 67).

Protect the User and Role Collections MongoDB stores role and user data in the protected `admin.system.roles` and `admin.system.users` collections, which are only accessible using the *user management methods*.

If you disable access control, **do not** modify the `admin.system.roles` and `admin.system.users` collections using normal `insert()` and `update()` operations.

Additional Information

See the reference section for documentation of all *built-in-roles* (page 85) and all available *privilege actions* (page 99). Also consider the reference for the form of the *resource documents* (page 97).

To create users see the *Create a User Administrator* (page 67) and *Manage User and Roles* (page 69) tutorials.

Collection-Level Access Control

Collection-level access control allows administrators to grant users privileges that are scoped to specific collections.

Administrators can implement collection-level access control through *user-defined roles* (page 15). By creating a role with *privileges* (page 15) that are scoped to a specific collection in a particular database, administrators can provision users with roles that grant privileges on a collection level.

Privileges and Scope

A privilege consists of *actions* (page 99) and the *resources* (page 97) upon which the actions are permissible; i.e. the resources define the scope of the actions for that privilege.

By specifying both the database and the collection in the *resource document* (page 97) for a privilege, administrator can limit the privilege actions just to a specific collection in a specific database. Each privilege action in a role can be scoped to a different collection.

For example, a user defined role can contain the following privileges:

```
privileges: [
  { resource: { db: "products", collection: "inventory" }, actions: [ "find", "update", "insert" ] },
  { resource: { db: "products", collection: "orders" },  actions: [ "find" ] }
]
```

The first privilege scopes its actions to the `inventory` collection of the `products` database. The second privilege scopes its actions to the `orders` collection of the `products` database.

Additional Information

For more information on user-defined roles and MongoDB authorization model, see *Authorization* (page 14). For a tutorial on creating user-defined roles, see *Manage User and Roles* (page 69).

2.3 Auditing

New in version 2.6.

MongoDB Enterprise includes an auditing capability for `mongod` and `mongos` instances. The auditing facility allows administrators and users to track system activity for deployments with multiple users and applications. The auditing facility can write audit events to the console, the *syslog*, a JSON file, or a BSON file.

Audit Events and Filter

To enable auditing for MongoDB Enterprise, see *Configure System Events Auditing* (page 78).

Once enabled, the auditing system can record the following operations:

- schema (DDL),

- replica set,
- authentication and authorization, and
- general operations.

For details on the audit log messages, see *System Event Audit Messages* (page 104).

By default, the auditing system records all these operations; however, you can *set up filters* (page 80) to restrict the events captured. To set up filters, see *Configure Audit Filters* (page 80).

Audit Guarantee

The auditing system writes every audit event [29] to an in-memory buffer of audit events. MongoDB writes this buffer to disk periodically. For events collected from any single connection, the events have a total order: if MongoDB writes one event to disk, the system guarantees that it has written all prior events for that connection to disk.

If an audit event entry corresponds to an operation that affects the durable state of the database, such as a modification to data, MongoDB will always write the audit event to disk *before* writing to the *journal* for that entry.

That is, before adding an operation to the journal, MongoDB writes all audit events on the connection that triggered the operation, up to and including the entry for the operation.

These auditing guarantees require that MongoDB run with `journaling` enabled.

> **Warning:** MongoDB may lose events **if** the server terminates before it commits the events to the audit log. The client may receive confirmation of the event before MongoDB commits to the audit log. For example, while auditing an aggregation operation, the server might crash after returning the result but before the audit log flushes.

2.4 External Environment

These documents introduce and address concepts and strategies related to security practices in MongoDB deployments.

Security and MongoDB API Interfaces **(page 17)** Discusses potential risks related to MongoDB's JavaScript, HTTP and REST interfaces, including strategies to control those risks.

Security and MongoDB API Interfaces

The following section contains strategies to limit risks related to MongoDB's available interfaces including JavaScript, HTTP, and REST interfaces.

JavaScript and the Security of the `mongo` Shell

The following JavaScript evaluation behaviors of the `mongo` shell represents risk exposures.

JavaScript Expression or JavaScript File The `mongo` program can evaluate JavaScript expressions using the command line `--eval` option. Also, the `mongo` program can evaluate a JavaScript file (`.js`) passed directly to it (e.g. `mongo someFile.js`).

Because the `mongo` program evaluates the JavaScript directly, inputs should only come from trusted sources.

[29] Audit configuration can include a *filter* (page 80) to limit events to audit.

`.mongorc.js` File If a `.mongorc.js` file exists [30], the `mongo` shell will evaluate a `.mongorc.js` file before starting. You can disable this behavior by passing the *`mongo --norc`* option.

HTTP Status Interface

Warning: Ensure that the HTTP status interface, the REST API, and the JSON API are all disabled in production environments to prevent potential data exposure and vulnerability to attackers.

The HTTP status interface provides a web-based interface that includes a variety of operational data, logs, and status reports regarding the `mongod` or `mongos` instance. The HTTP interface is always available on the port numbered `1000` greater than the primary `mongod` port. By default, the HTTP interface port is `28017`, but is indirectly set using the `port` option which allows you to configure the primary `mongod` port.

Without the `net.http.RESTInterfaceEnabled` setting, this interface is entirely read-only, and limited in scope; nevertheless, this interface may represent an exposure. To disable the HTTP interface, set the `enabled` run time option or the *`--nohttpinterface`* command line option. See also *Configuration Options* (page 10).

Note: While MongoDB Enterprise does support Kerberos authentication, Kerberos is not supported in HTTP status interface in any version of MongoDB.

Changed in version 3.0.

Neither the HTTP status interface nor the REST API support the *SCRAM-SHA-1* (page 6) challenge-response user authentication mechanism introduced in version 3.0.

REST API

The REST API to MongoDB provides additional information and write access on top of the HTTP status interface. While the REST API does not provide any support for insert, update, or remove operations, it does provide administrative access, and its accessibility represents a vulnerability in a secure environment. The REST interface is *disabled* by default, and is not recommended for production use.

If you must use the REST API, please control and limit access to the REST API. The REST API does not include any support for authentication, even when running with `authorization` enabled.

See the following documents for instructions on restricting access to the REST API interface:

- *Configure Linux iptables Firewall for MongoDB* (page 20)
- *Configure Windows netsh Firewall for MongoDB* (page 23)

3 Security Tutorials

The following tutorials provide instructions for enabling and using the security features available in MongoDB.

Network Security Tutorials **(page 19)** Ensure that the underlying network configuration supports a secure operating environment for MongoDB deployments, and appropriately limits access to MongoDB deployments.

Configure Linux iptables Firewall for MongoDB **(page 20)** Basic firewall configuration patterns and examples for `iptables` on Linux systems.

[30] On Linux and Unix systems, `mongo` reads the `.mongorc.js` file from *`$HOME`*`/.mongorc.js` (i.e. `~/.mongorc.js`). On Windows, `mongo.exe` reads the `.mongorc.js` file from *`%HOME%`*`.mongorc.js` or *`%HOMEDRIVE%%HOMEPATH%`*`.mongorc.js`.

Configure Windows netsh Firewall for MongoDB **(page 23)** Basic firewall configuration patterns and examples for `netsh` on Windows systems.

Configure mongod and mongos for TLS/SSL **(page 26)** TLS/SSL allows MongoDB clients to support encrypted connections to `mongod` instances.

Continue reading from *Network Security Tutorials* (page 19) for more information on running MongoDB in secure environments.

Security Deployment Tutorials **(page 36)** These tutorials describe procedures for deploying MongoDB using authentication and authorization.

Authentication Tutorials **(page 39)** These tutorials describe procedures relevant for the configuration, operation, and maintenance of MongoDB's access control system.

Enable Client Access Control **(page 40)** Describes the process for enabling authentication for MongoDB deployments.

Use x.509 Certificates to Authenticate Clients **(page 44)** Use x.509 for client authentication.

Use x.509 Certificate for Membership Authentication **(page 46)** Use x.509 for internal member authentication for replica sets and sharded clusters.

Configure MongoDB with Kerberos Authentication on Linux **(page 55)** For MongoDB Enterprise Linux, describes the process to enable Kerberos-based authentication for MongoDB deployments.

Continue reading from *Authentication Tutorials* (page 39) for additional tutorials on configuring MongoDB's authentication systems.

Enable Authentication after Creating the User Administrator **(page 42)** Describes an alternative process for enabling authentication for MongoDB deployments.

User and Role Management Tutorials **(page 66)** MongoDB's access control system provides a flexible role-based access control system that you can use to limit access to MongoDB deployments. The tutorials in this section describe the configuration an setup of the authorization system.

Manage User and Roles **(page 69)** Manage users by creating new users, creating new roles, and modifying existing users.

Continue reading from *User and Role Management Tutorials* (page 66) for additional tutorials on managing users and privileges in MongoDB's authorization system.

Auditing Tutorials **(page 78)** MongoDB Enterprise provides auditing of operations. The tutorials in this section describe procedures to enable and configure the auditing feature.

Create a Vulnerability Report **(page 83)** Report a vulnerability in MongoDB.

3.1 Network Security Tutorials

The following tutorials provide information on handling network security for MongoDB.

Configure Linux iptables Firewall for MongoDB **(page 20)** Basic firewall configuration patterns and examples for `iptables` on Linux systems.

Configure Windows netsh Firewall for MongoDB **(page 23)** Basic firewall configuration patterns and examples for `netsh` on Windows systems.

Configure mongod and mongos for TLS/SSL **(page 26)** TLS/SSL allows MongoDB clients to support encrypted connections to `mongod` instances.

TLS/SSL Configuration for Clients **(page 30)** Configure clients to connect to MongoDB instances that use TLS/SSL.

Upgrade a Cluster to Use TLS/SSL **(page 33)** Rolling upgrade process to use TLS/SSL.

Configure MongoDB for FIPS **(page 34)** Configure for Federal Information Processing Standard (FIPS).

Configure Linux `iptables` Firewall for MongoDB

On contemporary Linux systems, the `iptables` program provides methods for managing the Linux Kernel's `netfilter` or network packet filtering capabilities. These firewall rules make it possible for administrators to control what hosts can connect to the system, and limit risk exposure by limiting the hosts that can connect to a system.

This document outlines basic firewall configurations for `iptables` firewalls on Linux. Use these approaches as a starting point for your larger networking organization. For a detailed overview of security practices and risk management for MongoDB, see *Security Concepts* (page 5).

See also:

For MongoDB deployments on Amazon's web services, see the Amazon EC2[31] page, which addresses Amazon's Security Groups and other EC2-specific security features.

Overview

Rules in `iptables` configurations fall into chains, which describe the process for filtering and processing specific streams of traffic. Chains have an order, and packets must pass through earlier rules in a chain to reach later rules. This document addresses only the following two chains:

`INPUT` Controls all incoming traffic.

`OUTPUT` Controls all outgoing traffic.

Given the *default ports* (page 10) of all MongoDB processes, you must configure networking rules that permit *only* required communication between your application and the appropriate `mongod` and `mongos` instances.

Be aware that, by default, the default policy of `iptables` is to allow all connections and traffic unless explicitly disabled. The configuration changes outlined in this document will create rules that explicitly allow traffic from specific addresses and on specific ports, using a default policy that drops all traffic that is not explicitly allowed. When you have properly configured your `iptables` rules to allow only the traffic that you want to permit, you can *Change Default Policy to DROP* (page 22).

Patterns

This section contains a number of patterns and examples for configuring `iptables` for use with MongoDB deployments. If you have configured different ports using the `port` configuration setting, you will need to modify the rules accordingly.

Traffic to and from `mongod` Instances This pattern is applicable to all `mongod` instances running as standalone instances or as part of a *replica set*.

The goal of this pattern is to explicitly allow traffic to the `mongod` instance from the application server. In the following examples, replace `<ip-address>` with the IP address of the application server:

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 27017 -m state --state NEW,ESTABLISHED -j
iptables -A OUTPUT -d <ip-address> -p tcp --source-port 27017 -m state --state ESTABLISHED -j ACCEPT
```

[31] https://docs.mongodb.org/ecosystem/platforms/amazon-ec2

The first rule allows all incoming traffic from <ip-address> on port 27017, which allows the application server to connect to the mongod instance. The second rule, allows outgoing traffic from the mongod to reach the application server.

Optional

If you have only one application server, you can replace <ip-address> with either the IP address itself, such as: 198.51.100.55. You can also express this using CIDR notation as 198.51.100.55/32. If you want to permit a larger block of possible IP addresses you can allow traffic from a /24 using one of the following specifications for the <ip-address>, as follows:

```
10.10.10.10/24
10.10.10.10/255.255.255.0
```

Traffic to and from `mongos` Instances mongos instances provide query routing for *sharded clusters*. Clients connect to mongos instances, which behave from the client's perspective as mongod instances. In turn, the mongos connects to all mongod instances that are components of the sharded cluster.

Use the same iptables command to allow traffic to and from these instances as you would from the mongod instances that are members of the replica set. Take the configuration outlined in the *Traffic to and from mongod Instances* (page 20) section as an example.

Traffic to and from a MongoDB Config Server Config servers, host the *config database* that stores metadata for sharded clusters. Each production cluster has three config servers, initiated using the *mongod --configsvr* option. [32] Config servers listen for connections on port 27019. As a result, add the following iptables rules to the config server to allow incoming and outgoing connection on port 27019, for connection to the other config servers.

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 27019 -m state --state NEW,ESTABLISHED -
iptables -A OUTPUT -d <ip-address> -p tcp --source-port 27019 -m state --state ESTABLISHED -j ACCEPT
```

Replace <ip-address> with the address or address space of *all* the mongod that provide config servers.

Additionally, config servers need to allow incoming connections from all of the mongos instances in the cluster *and* all mongod instances in the cluster. Add rules that resemble the following:

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 27019 -m state --state NEW,ESTABLISHED -
```

Replace <ip-address> with the address of the mongos instances and the shard mongod instances.

Traffic to and from a MongoDB Shard Server For shard servers, running as *mongod --shardsvr* [33] Because the default port number is 27018 when running with the shardsvr value for the clusterRole setting, you must configure the following iptables rules to allow traffic to and from each shard:

```
iptables -A INPUT -s <ip-address> -p tcp --destination-port 27018 -m state --state NEW,ESTABLISHED -
iptables -A OUTPUT -d <ip-address> -p tcp --source-port 27018 -m state --state ESTABLISHED -j ACCEPT
```

Replace the <ip-address> specification with the IP address of all mongod. This allows you to permit incoming and outgoing traffic between all shards including constituent replica set members, to:

- all mongod instances in the shard's replica sets.
- all mongod instances in other shards. [34]

[32] You also can run a config server by using the configsvr value for the clusterRole setting in a configuration file.

[33] You can also specify the shard server option with the shardsvr value for the clusterRole setting in the configuration file. Shard members are also often conventional replica sets using the default port.

[34] All shards in a cluster need to be able to communicate with all other shards to facilitate *chunk* and balancing operations.

Furthermore, shards need to be able make outgoing connections to:

- all `mongos` instances.
- all `mongod` instances in the config servers.

Create a rule that resembles the following, and replace the `<ip-address>` with the address of the config servers and the `mongos` instances:

```
iptables -A OUTPUT -d <ip-address> -p tcp --source-port 27018 -m state --state ESTABLISHED -j ACCEPT
```

Provide Access For Monitoring Systems

1. The `mongostat` diagnostic tool, when running with the *`--discover`* needs to be able to reach all components of a cluster, including the config servers, the shard servers, and the `mongos` instances.
2. If your monitoring system needs access the HTTP interface, insert the following rule to the chain:

   ```
   iptables -A INPUT -s <ip-address> -p tcp --destination-port 28017 -m state --state NEW,ESTABLISH
   ```

   Replace `<ip-address>` with the address of the instance that needs access to the HTTP or REST interface. For *all* deployments, you should restrict access to this port to *only* the monitoring instance.

   Optional

   For config server `mongod` instances running with the `shardsvr` value for the `clusterRole` setting, the rule would resemble the following:

   ```
   iptables -A INPUT -s <ip-address> -p tcp --destination-port 28018 -m state --state NEW,ESTABLISH
   ```

   For config server `mongod` instances running with the `configsvr` value for the `clusterRole` setting, the rule would resemble the following:

   ```
   iptables -A INPUT -s <ip-address> -p tcp --destination-port 28019 -m state --state NEW,ESTABLISH
   ```

Change Default Policy to `DROP`

The default policy for `iptables` chains is to allow all traffic. After completing all `iptables` configuration changes, you *must* change the default policy to `DROP` so that all traffic that isn't explicitly allowed as above will not be able to reach components of the MongoDB deployment. Issue the following commands to change this policy:

```
iptables -P INPUT DROP

iptables -P OUTPUT DROP
```

Manage and Maintain `iptables` Configuration

This section contains a number of basic operations for managing and using `iptables`. There are various front end tools that automate some aspects of `iptables` configuration, but at the core all `iptables` front ends provide the same basic functionality:

Make all `iptables` Rules Persistent By default all `iptables` rules are only stored in memory. When your system restarts, your firewall rules will revert to their defaults. When you have tested a rule set and have guaranteed that it effectively controls traffic you can use the following operations to you should make the rule set persistent.

On Red Hat Enterprise Linux, Fedora Linux, and related distributions you can issue the following command:

```
service iptables save
```

On Debian, Ubuntu, and related distributions, you can use the following command to dump the `iptables` rules to the `/etc/iptables.conf` file:

```
iptables-save > /etc/iptables.conf
```

Run the following operation to restore the network rules:

```
iptables-restore < /etc/iptables.conf
```

Place this command in your `rc.local` file, or in the `/etc/network/if-up.d/iptables` file with other similar operations.

List all `iptables` Rules To list all of currently applied `iptables` rules, use the following operation at the system shell.

```
iptables -L
```

Flush all `iptables` Rules If you make a configuration mistake when entering `iptables` rules or simply need to revert to the default rule set, you can use the following operation at the system shell to flush all rules:

```
iptables -F
```

If you've already made your `iptables` rules persistent, you will need to repeat the appropriate procedure in the *Make all iptables Rules Persistent* (page 22) section.

Configure Windows `netsh` Firewall for MongoDB

On Windows Server systems, the `netsh` program provides methods for managing the *Windows Firewall*. These firewall rules make it possible for administrators to control what hosts can connect to the system, and limit risk exposure by limiting the hosts that can connect to a system.

This document outlines basic *Windows Firewall* configurations. Use these approaches as a starting point for your larger networking organization. For a detailed over view of security practices and risk management for MongoDB, see *Security Concepts* (page 5).

See also:

Windows Firewall[35] documentation from Microsoft.

Overview

Windows Firewall processes rules in an ordered determined by rule type, and parsed in the following order:

1. `Windows Service Hardening`
2. `Connection security rules`
3. `Authenticated Bypass Rules`
4. `Block Rules`
5. `Allow Rules`

[35] http://technet.microsoft.com/en-us/network/bb545423.aspx

6. Default Rules

By default, the policy in *Windows Firewall* allows all outbound connections and blocks all incoming connections.

Given the *default ports* (page 10) of all MongoDB processes, you must configure networking rules that permit *only* required communication between your application and the appropriate `mongod.exe` and `mongos.exe` instances.

The configuration changes outlined in this document will create rules which explicitly allow traffic from specific addresses and on specific ports, using a default policy that drops all traffic that is not explicitly allowed.

You can configure the *Windows Firewall* with using the `netsh` command line tool or through a windows application. On Windows Server 2008 this application is *Windows Firewall With Advanced Security* in *Administrative Tools*. On previous versions of Windows Server, access the *Windows Firewall* application in the *System and Security* control panel.

The procedures in this document use the `netsh` command line tool.

Patterns

This section contains a number of patterns and examples for configuring *Windows Firewall* for use with MongoDB deployments. If you have configured different ports using the `port` configuration setting, you will need to modify the rules accordingly.

Traffic to and from `mongod.exe` Instances This pattern is applicable to all `mongod.exe` instances running as standalone instances or as part of a *replica set*. The goal of this pattern is to explicitly allow traffic to the `mongod.exe` instance from the application server.

```
netsh advfirewall firewall add rule name="Open mongod port 27017" dir=in action=allow protocol=TCP lo
```

This rule allows all incoming traffic to port `27017`, which allows the application server to connect to the `mongod.exe` instance.

Windows Firewall also allows enabling network access for an entire application rather than to a specific port, as in the following example:

```
netsh advfirewall firewall add rule name="Allowing mongod" dir=in action=allow program=" C:\mongodb\b
```

You can allow all access for a `mongos.exe` server, with the following invocation:

```
netsh advfirewall firewall add rule name="Allowing mongos" dir=in action=allow program=" C:\mongodb\b
```

Traffic to and from `mongos.exe` Instances `mongos.exe` instances provide query routing for *sharded clusters*. Clients connect to `mongos.exe` instances, which behave from the client's perspective as `mongod.exe` instances. In turn, the `mongos.exe` connects to all `mongod.exe` instances that are components of the sharded cluster.

Use the same *Windows Firewall* command to allow traffic to and from these instances as you would from the `mongod.exe` instances that are members of the replica set.

```
netsh advfirewall firewall add rule name="Open mongod shard port 27018" dir=in action=allow protocol=
```

Traffic to and from a MongoDB Config Server Configuration servers, host the *config database* that stores metadata for sharded clusters. Each production cluster has three configuration servers, initiated using the *`mongod --configsvr`* option. [36] Configuration servers listen for connections on port `27019`. As a result, add the following *Windows Firewall* rules to the config server to allow incoming and outgoing connection on port `27019`, for connection to the other config servers.

[36] You also can run a config server by using the `configsrv` value for the `clusterRole` setting in a configuration file.

```
netsh advfirewall firewall add rule name="Open mongod config svr port 27019" dir=in action=allow prot
```

Additionally, config servers need to allow incoming connections from all of the `mongos.exe` instances in the cluster *and* all `mongod.exe` instances in the cluster. Add rules that resemble the following:

```
netsh advfirewall firewall add rule name="Open mongod config svr inbound" dir=in action=allow protoco
```

Replace `<ip-address>` with the addresses of the `mongos.exe` instances and the shard `mongod.exe` instances.

Traffic to and from a MongoDB Shard Server For shard servers, running as *`mongod --shardsvr`* [37] Because the default port number is `27018` when running with the `shardsvr` value for the `clusterRole` setting, you must configure the following *Windows Firewall* rules to allow traffic to and from each shard:

```
netsh advfirewall firewall add rule name="Open mongod shardsvr inbound" dir=in action=allow protocol=
netsh advfirewall firewall add rule name="Open mongod shardsvr outbound" dir=out action=allow protoco
```

Replace the `<ip-address>` specification with the IP address of all `mongod.exe` instances. This allows you to permit incoming and outgoing traffic between all shards including constituent replica set members to:

- all `mongod.exe` instances in the shard's replica sets.
- all `mongod.exe` instances in other shards. [38]

Furthermore, shards need to be able make outgoing connections to:

- all `mongos.exe` instances.
- all `mongod.exe` instances in the config servers.

Create a rule that resembles the following, and replace the `<ip-address>` with the address of the config servers and the `mongos.exe` instances:

```
netsh advfirewall firewall add rule name="Open mongod config svr outbound" dir=out action=allow proto
```

Provide Access For Monitoring Systems

1. The `mongostat` diagnostic tool, when running with the *`--discover`* needs to be able to reach all components of a cluster, including the config servers, the shard servers, and the `mongos.exe` instances.
2. If your monitoring system needs access the HTTP interface, insert the following rule to the chain:

   ```
   netsh advfirewall firewall add rule name="Open mongod HTTP monitoring inbound" dir=in action=all
   ```

   Replace `<ip-address>` with the address of the instance that needs access to the HTTP or REST interface. For *all* deployments, you should restrict access to this port to *only* the monitoring instance.

   Optional

   For config server `mongod` instances running with the `shardsvr` value for the `clusterRole` setting, the rule would resemble the following:

   ```
   netsh advfirewall firewall add rule name="Open mongos HTTP monitoring inbound" dir=in action=all
   ```

   For config server `mongod` instances running with the `configsvr` value for the `clusterRole` setting, the rule would resemble the following:

[37] You can also specify the shard server option with the `shardsvr` value for the `clusterRole` setting in the configuration file. Shard members are also often conventional replica sets using the default port.

[38] All shards in a cluster need to be able to communicate with all other shards to facilitate *chunk* and balancing operations.

```
netsh advfirewall firewall add rule name="Open mongod configsvr HTTP monitoring inbound" dir=in
```

Manage and Maintain *Windows Firewall* Configurations

This section contains a number of basic operations for managing and using `netsh`. While you can use the GUI front ends to manage the *Windows Firewall*, all core functionality is accessible is accessible from `netsh`.

Delete all *Windows Firewall* Rules To delete the firewall rule allowing `mongod.exe` traffic:

```
netsh advfirewall firewall delete rule name="Open mongod port 27017" protocol=tcp localport=27017
```

```
netsh advfirewall firewall delete rule name="Open mongod shard port 27018" protocol=tcp localport=270
```

List All *Windows Firewall* Rules To return a list of all *Windows Firewall* rules:

```
netsh advfirewall firewall show rule name=all
```

Reset *Windows Firewall* To reset the *Windows Firewall* rules:

```
netsh advfirewall reset
```

Backup and Restore *Windows Firewall* Rules To simplify administration of larger collection of systems, you can export or import firewall systems from different servers) rules very easily on Windows:

Export all firewall rules with the following command:

```
netsh advfirewall export "C:\temp\MongoDBfw.wfw"
```

Replace `"C:\temp\MongoDBfw.wfw"` with a path of your choosing. You can use a command in the following form to import a file created using this operation:

```
netsh advfirewall import "C:\temp\MongoDBfw.wfw"
```

Configure `mongod` and `mongos` for TLS/SSL

Overview

This document helps you to configure MongoDB to support TLS/SSL. MongoDB clients can use TLS/SSL to encrypt connections to `mongod` and `mongos` instances. MongoDB TLS/SSL implementation uses OpenSSL libraries.

Note: Although TLS is the successor to SSL, this page uses the more familiar term SSL to refer to TLS/SSL.

These instructions assume that you have already installed a build of MongoDB that includes SSL support and that your client driver supports SSL. For instructions on upgrading a cluster currently not using SSL to using SSL, see *Upgrade a Cluster to Use TLS/SSL* (page 33).

Changed in version 2.6: MongoDB's SSL encryption only allows use of strong SSL ciphers with a minimum of 128-bit key length for all connections.

Prerequisites

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, and Certificate Authority is beyond the scope of this document. This page assumes prior knowledge of TLS/SSL as well as access to valid certificates.

MongoDB Support New in version 3.0: Most MongoDB distributions now include support for SSL.

Certain distributions of MongoDB[39] do **not** contain support for SSL. To use SSL, be sure to choose a package that supports SSL. All MongoDB Enterprise[40] supported platforms include SSL support.

Client Support See *TLS/SSL Configuration for Clients* (page 30) to learn about SSL support for Python, Java, Ruby, and other clients.

Certificate Authorities For production use, your MongoDB deployment should use valid certificates generated and signed by a single certificate authority. You or your organization can generate and maintain an independent certificate authority, or use certificates generated by a third-party SSL vendor. Obtaining and managing certificates is beyond the scope of this documentation.

`.pem` File Before you can use SSL, you must have a `.pem` file containing a public key certificate and its associated private key.

MongoDB can use any valid SSL certificate issued by a certificate authority, or a self-signed certificate. If you use a self-signed certificate, although the communications channel will be encrypted, there will be *no* validation of server identity. Although such a situation will prevent eavesdropping on the connection, it leaves you vulnerable to a man-in-the-middle attack. Using a certificate signed by a trusted certificate authority will permit MongoDB drivers to verify the server's identity.

In general, avoid using self-signed certificates unless the network is trusted.

Additionally, with regards to *authentication among replica set/sharded cluster members* (page 8), in order to minimize exposure of the private key and allow hostname validation, it is advisable to use different certificates on different servers.

For *testing* purposes, you can generate a self-signed certificate and private key on a Unix system with a command that resembles the following:

```
cd /etc/ssl/
openssl req -newkey rsa:2048 -new -x509 -days 365 -nodes -out mongodb-cert.crt -keyout mongodb-cert.k
```

This operation generates a new, self-signed certificate with no passphrase that is valid for 365 days. Once you have the certificate, concatenate the certificate and private key to a `.pem` file, as in the following example:

```
cat mongodb-cert.key mongodb-cert.crt > mongodb.pem
```

See also:

Use x.509 Certificates to Authenticate Clients (page 44)

[39] http://www.mongodb.org/downloads?jmp=docs
[40] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs

Procedures

Set Up `mongod` and `mongos` with SSL Certificate and Key To use SSL in your MongoDB deployment, include the following run-time options with `mongod` and `mongos`:

- `net.ssl.mode` set to `requireSSL`. This setting restricts each server to use only SSL encrypted connections. You can also specify either the value `allowSSL` or `preferSSL` to set up the use of mixed SSL modes on a port. See `net.ssl.mode` for details.
- `PEMKeyfile` with the `.pem` file that contains the SSL certificate and key.

Consider the following syntax for `mongod`:

```
mongod --sslMode requireSSL --sslPEMKeyFile <pem>
```

For example, given an SSL certificate located at `/etc/ssl/mongodb.pem`, configure `mongod` to use SSL encryption for all connections with the following command:

```
mongod --sslMode requireSSL --sslPEMKeyFile /etc/ssl/mongodb.pem
```

Note:

- Specify `<pem>` with the full path name to the certificate.
- If the private key portion of the `<pem>` is encrypted, specify the passphrase. See *SSL Certificate Passphrase* (page 30).

You may also specify these options in the `configuration file`, as in the following examples:

If using the `YAML configuration file format`:

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

Or, if using the older older configuration file format[41]:

```
sslMode = requireSSL
sslPEMKeyFile = /etc/ssl/mongodb.pem
```

To connect, to `mongod` and `mongos` instances using SSL, the `mongo` shell and MongoDB tools must include the `--ssl` option. See *TLS/SSL Configuration for Clients* (page 30) for more information on connecting to `mongod` and `mongos` running with SSL.

See also:

Upgrade a Cluster to Use TLS/SSL (page 33)

Set Up `mongod` and `mongos` with Certificate Validation To set up `mongod` or `mongos` for SSL encryption using an SSL certificate signed by a certificate authority, include the following run-time options during startup:

- `net.ssl.mode` set to `requireSSL`. This setting restricts each server to use only SSL encrypted connections. You can also specify either the value `allowSSL` or `preferSSL` to set up the use of mixed SSL modes on a port. See `net.ssl.mode` for details.
- `PEMKeyfile` with the name of the `.pem` file that contains the signed SSL certificate and key.
- `CAFile` with the name of the `.pem` file that contains the root certificate chain from the Certificate Authority.

[41] http://docs.mongodb.org/v2.4/reference/configuration-options

Consider the following syntax for `mongod`:

```
mongod --sslMode requireSSL --sslPEMKeyFile <pem> --sslCAFile <ca>
```

For example, given a signed SSL certificate located at `/etc/ssl/mongodb.pem` and the certificate authority file at `/etc/ssl/ca.pem`, you can configure `mongod` for SSL encryption as follows:

```
mongod --sslMode requireSSL --sslPEMKeyFile /etc/ssl/mongodb.pem --sslCAFile /etc/ssl/ca.pem
```

Note:

- Specify the `<pem>` file and the `<ca>` file with either the full path name or the relative path name.
- If the `<pem>` is encrypted, specify the passphrase. See *SSL Certificate Passphrase* (page 30).

You may also specify these options in the `configuration file`, as in the following examples:

If using the `YAML configuration file format`:

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

Or, if using the older older configuration file format[42]:

```
sslMode = requireSSL
sslPEMKeyFile = /etc/ssl/mongodb.pem
sslCAFile = /etc/ssl/ca.pem
```

To connect, to `mongod` and `mongos` instances using SSL, the `mongo` tools must include the both the *`--ssl`* and *`--sslPEMKeyFile`* option. See *TLS/SSL Configuration for Clients* (page 30) for more information on connecting to `mongod` and `mongos` running with SSL.

See also:

Upgrade a Cluster to Use TLS/SSL (page 33)

Block Revoked Certificates for Clients To prevent clients with revoked certificates from connecting, include the `sslCRLFile` to specify a `.pem` file that contains revoked certificates.

For example, the following `mongod` with SSL configuration includes the `sslCRLFile` setting:

```
mongod --sslMode requireSSL --sslCRLFile /etc/ssl/ca-crl.pem --sslPEMKeyFile /etc/ssl/mongodb.pem --s
```

Clients with revoked certificates in the `/etc/ssl/ca-crl.pem` will not be able to connect to this `mongod` instance.

Validate Only if a Client Presents a Certificate In most cases it is important to ensure that clients present valid certificates. However, if you have clients that cannot present a client certificate, or are transitioning to using a certificate authority you may only want to validate certificates from clients that present a certificate.

If you want to bypass validation for clients that don't present certificates, include the `allowConnectionsWithoutCertificates` run-time option with `mongod` and `mongos`. If the client does not present a certificate, no validation occurs. These connections, though not validated, are still encrypted using SSL.

[42] http://docs.mongodb.org/v2.4/reference/configuration-options

For example, consider the following `mongod` with an SSL configuration that includes the `allowConnectionsWithoutCertificates` setting:

```
mongod --sslMode requireSSL --sslAllowConnectionsWithoutCertificates --sslPEMKeyFile /etc/ssl/mongodb
```

Then, clients can connect either with the option *`--ssl`* and **no** certificate or with the option *`--ssl`* and a **valid** certificate. See *TLS/SSL Configuration for Clients* (page 30) for more information on SSL connections for clients.

Note: If the client presents a certificate, the certificate must be a valid certificate.

All connections, including those that have not presented certificates are encrypted using SSL.

SSL Certificate Passphrase The PEM files for `PEMKeyfile` and `ClusterFile` may be encrypted. With encrypted PEM files, you must specify the passphrase at startup with a command-line or a configuration file option or enter the passphrase when prompted.

Changed in version 2.6: In previous versions, you can only specify the passphrase with a command-line or a configuration file option.

To specify the passphrase in clear text on the command line or in a configuration file, use the `PEMKeyPassword` and/or the `ClusterPassword` option.

To have MongoDB prompt for the passphrase at the start of `mongod` or `mongos` and avoid specifying the passphrase in clear text, omit the `PEMKeyPassword` and/or the `ClusterPassword` option. MongoDB will prompt for each passphrase as necessary.

Important: The passphrase prompt option is available if you run the MongoDB instance in the foreground with a connected terminal. If you run `mongod` or `mongos` in a non-interactive session (e.g. without a terminal or as a service on Windows), you cannot use the passphrase prompt option.

Run in FIPS Mode

Note: FIPS-compatible SSL is available only in MongoDB Enterprise[43]. See *Configure MongoDB for FIPS* (page 34) for more information.

See *Configure MongoDB for FIPS* (page 34) for more details.

TLS/SSL Configuration for Clients

Clients must have support for TLS/SSL to work with a `mongod` or a `mongos` instance that has TLS/SSL support enabled.

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, and Certificate Authority is beyond the scope of this document. This page assumes prior knowledge of TLS/SSL as well as access to valid certificates.

Note: Although TLS is the successor to SSL, this page uses the more familiar term SSL to refer to TLS/SSL.

See also:

Configure mongod and mongos for TLS/SSL (page 26).

[43] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs

mongo Shell SSL Configuration

For SSL connections, you must use the mongo shell built with SSL support or distributed with MongoDB Enterprise.

New in version 3.0: Most MongoDB distributions now include support for SSL.

The mongo shell provides various *mongo-shell-ssl* settings, including:

- *--ssl*
- *--sslPEMKeyFile* with the name of the .pem file that contains the SSL certificate and key.
- *--sslCAFile* with the name of the .pem file that contains the certificate from the Certificate Authority (CA).

  Changed in version 3.0: When running mongo with the --ssl option, you must include either --sslCAFile or --sslAllowInvalidCertificates.

  This restriction does not apply to the MongoDB tools. However, running the tools without –sslCAFile creates the same vulnerability to invalid certificates.

  > **Warning:** For SSL connections (--ssl) to mongod and mongos, if the mongo shell (or *MongoDB tools* (page 33)) runs without the --sslCAFile <CAFile> option (i.e. specifies the --sslAllowInvalidCertificates instead), the mongo shell (or *MongoDB tools* (page 33)) will not attempt to validate the server certificates. This creates a vulnerability to expired mongod and mongos certificates as well as to foreign processes posing as valid mongod or mongos instances. Ensure that you *always* specify the CA file to validate the server certificates in cases where intrusion is a possibility.

- *--sslPEMKeyPassword* option if the client certificate-key file is encrypted.

For a complete list of the mongo shell's SSL settings, see *mongo-shell-ssl*.

Connect to MongoDB Instance with SSL Encryption To connect to a mongod or mongos instance that requires *only a SSL encryption mode* (page 28), start mongo shell with *--ssl* and include the *--sslCAFile* to validate the server certificates.

```
mongo --ssl --sslCAFile /etc/ssl/ca.pem
```

Changed in version 3.0: When running mongo with the --ssl option, you must include either --sslCAFile or --sslAllowInvalidCertificates.

This restriction does not apply to the MongoDB tools. However, running the tools without –sslCAFile creates the same vulnerability to invalid certificates.

Connect to MongoDB Instance that Requires Client Certificates To connect to a mongod or mongos that requires *CA-signed client certificates* (page 28), start the mongo shell with *--ssl*, the *--sslPEMKeyFile* option to specify the signed certificate-key file, and the *--sslCAFile* to validate the server certificates.

```
mongo --ssl --sslPEMKeyFile /etc/ssl/client.pem --sslCAFile /etc/ssl/ca.pem
```

Changed in version 3.0: When running mongo with the --ssl option, you must include either --sslCAFile or --sslAllowInvalidCertificates.

This restriction does not apply to the MongoDB tools. However, running the tools without –sslCAFile creates the same vulnerability to invalid certificates.

Connect to MongoDB Instance that Validates when Presented with a Certificate To connect to a `mongod` or `mongos` instance that *only requires valid certificates when the client presents a certificate* (page 29), start `mongo` shell either:

- with the `--ssl`, `--sslCAFile`, and **no** certificate or
- with the `--ssl`, `--sslCAFile`, and a **valid** signed certificate.

Changed in version 3.0: When running `mongo` with the `--ssl` option, you must include either `--sslCAFile` or `--sslAllowInvalidCertificates`.

This restriction does not apply to the MongoDB tools. However, running the tools without `-sslCAFile` creates the same vulnerability to invalid certificates.

For example, if `mongod` is running with weak certificate validation, both of the following `mongo` shell clients can connect to that `mongod`:

```
mongo --ssl --sslCAFile /etc/ssl/ca.pem
mongo --ssl --sslPEMKeyFile /etc/ssl/client.pem --sslCAFile /etc/ssl/ca.pem
```

Important: If the client presents a certificate, the certificate must be valid.

MongoDB Cloud Manager and Ops Manager Monitoring Agent

The MongoDB Cloud Manager Monitoring agent will also have to connect via SSL in order to gather its statistics. Because the agent already utilizes SSL for its communications to the MongoDB Cloud Manager servers, this is just a matter of enabling SSL support in MongoDB Cloud Manager itself on a per host basis. å See the MongoDB Cloud Manager documentation[44] for more information about SSL configuration.

For Ops Manager, see Ops Manager documentation[45].

MongoDB Drivers

The MongoDB Drivers support for connection to SSL enabled MongoDB. See:

- C Driver[46]
- C++ Driver[47]
- C# Driver[48]
- Java Driver[49]
- Node.js Driver[50]
- Perl Driver[51]
- PHP Driver[52]
- Python Driver[53]

[44] https://docs.cloud.mongodb.com/
[45] https://docs.opsmanager.mongodb.com/current/
[46] http://api.mongodb.org/c/current/advanced-connections.html
[47] https://github.com/mongodb/mongo-cxx-driver/wiki/Configuring%20the%20Legacy%20Driver
[48] http://mongodb.github.io/mongo-csharp-driver/2.0/reference/driver/ssl/
[49] http://mongodb.github.io/mongo-java-driver/3.0/driver/reference/connecting/ssl/
[50] http://mongodb.github.io/node-mongodb-native/2.0/tutorials/enterprise_features/
[51] https://metacpan.org/pod/MongoDB::MongoClient#ssl
[52] http://php.net/manual/en/mongo.connecting.ssl.php
[53] http://api.mongodb.org/python/current/examples/tls.html

- Ruby Driver[54]
- Scala Driver[55]

MongoDB Tools

Changed in version 2.6.

Various MongoDB utility programs supports SSL. These tools include:

- `mongodump`
- `mongoexport`
- `mongofiles`
- `mongoimport`
- `mongooplog`
- `mongorestore`
- `mongostat`
- `mongotop`

To use SSL connections with these tools, use the same SSL options as the `mongo` shell. See *mongo Shell SSL Configuration* (page 31).

Upgrade a Cluster to Use TLS/SSL

Changed in version 3.0: Most MongoDB distributions now include support for TLS/SSL. See *Configure mongod and mongos for TLS/SSL* (page 26) and *TLS/SSL Configuration for Clients* (page 30) for more information about TLS/SSL and MongoDB.

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, and Certificate Authority is beyond the scope of this document. This page assumes prior knowledge of TLS/SSL as well as access to valid certificates.

Changed in version 2.6.

The MongoDB server supports listening for both TLS/SSL encrypted and unencrypted connections on the same TCP port. This allows upgrades of MongoDB clusters to use TLS/SSL encrypted connections.

To upgrade from a MongoDB cluster using no TLS/SSL encryption to one using *only* TLS/SSL encryption, use the following rolling upgrade process:

1. For each node of a cluster, start the node with the option `--sslMode` set to `allowSSL`. The `--sslMode allowSSL` setting allows the node to accept both TLS/SSL and non-TLS/non-SSL incoming connections. Its connections to other servers do not use TLS/SSL. Include other *TLS/SSL options* (page 26) as well as any other options that are required for your specific configuration. For example:

   ```
   mongod --replSet <name> --sslMode allowSSL --sslPEMKeyFile <path to TLS/SSL Certificate and key
   ```

   Upgrade all nodes of the cluster to these settings.

   You may also specify these options in the `configuration file`. If using a `YAML format configuration file`, specify the following settings in the file:

[54] http://docs.mongodb.org/ecosystem/tutorial/ruby-driver-tutorial/#mongodb-x509-mechanism
[55] http://mongodb.github.io/casbah/guide/connecting.html#ssl-connections

```
net:
   ssl:
      mode: <disabled|allowSSL|preferSSL|requireSSL>
      PEMKeyFile: <path to TLS/SSL certificate and key PEM file>
      CAFile: <path to root CA PEM file>
```

Or, if using the older configuration file format[56]:

```
sslMode = <disabled|allowSSL|preferSSL|requireSSL>
sslPEMKeyFile = <path to TLS/SSL certificate and key PEM file>
sslCAFile = <path to root CA PEM file>
```

2. Switch all clients to use TLS/SSL. See *TLS/SSL Configuration for Clients* (page 30).
3. For each node of a cluster, use the `setParameter` command to update the `sslMode` to `preferSSL`. [57] With `preferSSL` as its `net.ssl.mode`, the node accepts both TLS/SSL and non-TLS/non-SSL incoming connections, and its connections to other servers use TLS/SSL. For example:

   ```
   db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "preferSSL" } )
   ```

   Upgrade all nodes of the cluster to these settings.

   At this point, all connections should be using TLS/SSL.
4. For each node of the cluster, use the `setParameter` command to update the `sslMode` to `requireSSL`. [1] With `requireSSL` as its `net.ssl.mode`, the node will reject any non-TLS/non-SSL connections. For example:

   ```
   db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "requireSSL" } )
   ```

5. After the upgrade of all nodes, edit the `configuration file` with the appropriate TLS/SSL settings to ensure that upon subsequent restarts, the cluster uses TLS/SSL.

Configure MongoDB for FIPS

New in version 2.6.

Overview

The Federal Information Processing Standard (FIPS) is a U.S. government computer security standard used to certify software modules and libraries that encrypt and decrypt data securely. You can configure MongoDB to run with a FIPS 140-2 certified library for OpenSSL. Configure FIPS to run by default or as needed from the command line.

Prerequisites

Important: A full description of FIPS and TLS/SSL is beyond the scope of this document. This tutorial assumes prior knowledge of FIPS and TLS/SSL.

Only the MongoDB Enterprise[58] version supports FIPS mode. See `http://docs.mongodb.org/manual/administration/i` to download and install MongoDB Enterprise[59] to use FIPS mode.

[56] http://docs.mongodb.org/v2.4/reference/configuration-options
[57] As an alternative to using the `setParameter` command, you can also restart the nodes with the appropriate TLS/SSL options and values.
[58] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs
[59] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs

Your system must have an OpenSSL library configured with the FIPS 140-2 module. At the command line, type `openssl version` to confirm your OpenSSL software includes FIPS support.

For Red Hat Enterprise Linux 6.x (RHEL 6.x) or its derivatives such as CentOS 6.x, the OpenSSL toolkit must be at least `openssl-1.0.1e-16.el6_5` to use FIPS mode. To upgrade the toolkit for these platforms, issue the following command:

```
sudo yum update openssl
```

Some versions of Linux periodically execute a process to *prelink* dynamic libraries with pre-assigned addresses. This process modifies the OpenSSL libraries, specifically `libcrypto`. The OpenSSL FIPS mode will subsequently fail the signature check performed upon startup to ensure `libcrypto` has not been modified since compilation.

To configure the Linux prelink process to not prelink `libcrypto`:

```
sudo bash -c "echo '-b /usr/lib64/libcrypto.so.*' >>/etc/prelink.conf.d/openssl-prelink.conf"
```

Considerations

FIPS is property of the encryption system and not the access control system. However, if your environment requires FIPS compliant encryption *and* access control, you must ensure that the access control system uses only FIPS-compliant encryption.

MongoDB's FIPS support covers the way that MongoDB uses OpenSSL for network encryption and X509 authentication. If you use Kerberos or LDAP Proxy authentication, you muse ensure that these external mechanisms are FIPS-compliant. `MONGODB-CR` authentication is *not* FIPS compliant.

Procedure

Configure MongoDB to use TLS/SSL See *Configure mongod and mongos for TLS/SSL* (page 26) for details about configuring OpenSSL.

Run `mongod` or `mongos` instance in FIPS mode Perform these steps after you *Configure mongod and mongos for TLS/SSL* (page 26).

Step 1: Change configuration file. To configure your `mongod` or `mongos` instance to use FIPS mode, shut down the instance and update the configuration file with the following setting:

```
net:
   ssl:
      FIPSMode: true
```

Step 2: Start `mongod` or `mongos` instance with configuration file. For example, run this command to start the `mongod` instance with its configuration file:

```
mongod --config /etc/mongod.conf
```

Confirm FIPS mode is running Check the server log file for a message FIPS is active:

```
FIPS 140-2 mode activated
```

3.2 Security Deployment Tutorials

The following tutorials provide information in deploying MongoDB using authentication and authorization.

Deploy Replica Set and Configure Authentication and Authorization **(page 36)** Configure a replica set that has authentication enabled.

Deploy Replica Set and Configure Authentication and Authorization

Overview

With *authentication* (page 6) enabled, MongoDB forces all clients to identify themselves before granting access to the server. *Authorization* (page 14), in turn, allows administrators to define and limit the resources and operations that a user can access. Using authentication and authorization is a key part of a complete security strategy.

All MongoDB deployments support authentication. By default, MongoDB does not require authorization checking. You can enforce authorization checking when deploying MongoDB, or on an existing deployment; however, you cannot enable authorization checking on a running deployment without downtime.

This tutorial provides a procedure for creating a MongoDB `replica set` that uses the challenge-response authentication mechanism. The tutorial includes creation of a minimal authorization system to support basic operations.

Considerations

Authentication In this procedure, you will configure MongoDB using the default challenge-response authentication mechanism, using the `keyFile` to supply the password for *inter-process authentication* (page 8). The content of the key file is the shared secret used for all internal authentication.

All deployments that enforce authorization checking should have one *user administrator* user that can create new users and modify existing users. During this procedure you will create a user administrator that you will use to administer this deployment.

Architecture In a production, deploy each member of the replica set to its own machine and if possible bind to the standard MongoDB port of `27017`. Use the `bind_ip` option to ensure that MongoDB listens for connections from applications on configured addresses.

For a geographically distributed replica sets, ensure that the majority of the set's `mongod` instances reside in the primary site.

See `http://docs.mongodb.org/manual/core/replica-set-architectures` for more information.

Connectivity Ensure that network traffic can pass between all members of the set and all clients in the network securely and efficiently. Consider the following:

- Establish a virtual private network. Ensure that your network topology routes all traffic between members within a single site over the local area network.
- Configure access control to prevent connections from unknown clients to the replica set.
- Configure networking and firewall rules so that incoming and outgoing packets are permitted only on the default MongoDB port and only from within your deployment.

Finally ensure that each member of a replica set is accessible by way of resolvable DNS or hostnames. You should either configure your DNS names appropriately or set up your systems' `/etc/hosts` file to reflect this configuration.

Configuration Specify the run time configuration on each system in a `configuration file` stored in `/etc/mongod.conf` or a related location. Create the directory where MongoDB stores data files before deploying MongoDB.

For more information about the run time options used above and other configuration options, see `http://docs.mongodb.org/manual/reference/configuration-options`.

Procedure

This procedure deploys a replica set in which all members use the same key file.

Step 1: Start one member of the replica set. This `mongod` should *not* enable `auth`.

Step 2: Create administrative users. The following operations will create two users: a user administrator that will be able to create and modify users (`siteUserAdmin`), and a `root` (page 93) user (`siteRootAdmin`) that you will use to complete the remainder of the tutorial:

```
use admin
db.createUser( {
    user: "siteUserAdmin",
    pwd: "<password>",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  });
db.createUser( {
    user: "siteRootAdmin",
    pwd: "<password>",
    roles: [ { role: "root", db: "admin" } ]
  });
```

Step 3: Stop the `mongod` instance.

Step 4: Create the key file to be used by each member of the replica set. Create the key file your deployment will use to authenticate servers to each other.

To generate pseudo-random data to use for a `keyfile`, issue the following `openssl` command:

```
openssl rand -base64 741 > mongodb-keyfile
chmod 600 mongodb-keyfile
```

You may generate a key file using any method you choose. Always ensure that the password stored in the key file is both long and contains a high amount of entropy. Using `openssl` in this manner helps generate such a key.

Step 5: Copy the key file to each member of the replica set. Copy the `mongodb-keyfile` to all hosts where components of a MongoDB deployment run. Set the permissions of these files to `600` so that only the *owner* of the file can read or write this file to prevent other users on the system from accessing the shared secret.

Step 6: Start each member of the replica set with the appropriate options. For each member, start a `mongod` and specify the key file and the name of the replica set. Also specify other parameters as needed for your deployment. For replication-specific parameters, see *cli-mongod-replica-set* required by your deployment.

If your application connects to more than one replica set, each set should have a distinct name. Some drivers group replica set connections by replica set name.

The following example specifies parameters through the *--keyFile* and *--replSet* command-line options:

```
mongod --keyFile /mysecretdirectory/mongodb-keyfile --replSet "rs0"
```

The following example specifies parameters through a configuration file:

```
mongod --config $HOME/.mongodb/config
```

In production deployments, you can configure a *control script* to manage this process. Control scripts are beyond the scope of this document.

Step 7: Connect to the member of the replica set where you created the administrative users. Connect to the replica set member you started and authenticate as the `siteRootAdmin` user. From the `mongo` shell, use the following operation to authenticate:

```
use admin
db.auth("siteRootAdmin", "<password>");
```

Step 8: Initiate the replica set. Use `rs.initiate()` on the replica set member:

```
rs.initiate()
```

MongoDB initiates a set that consists of the current member and that uses the default replica set configuration.

Step 9: Verify the initial replica set configuration. Use `rs.conf()` to display the `replica set configuration object`:

```
rs.conf()
```

The replica set configuration object resembles the following:

```
{
   "_id" : "rs0",
   "version" : 1,
   "members" : [
      {
         "_id" : 1,
         "host" : "mongodb0.example.net:27017"
      }
   ]
}
```

Step 10: Add the remaining members to the replica set. Add the remaining members with the `rs.add()` method.

The following example adds two members:

```
rs.add("mongodb1.example.net")
rs.add("mongodb2.example.net")
```

When complete, you have a fully functional replica set. The new replica set will elect a *primary*.

Step 11: Check the status of the replica set. Use the `rs.status()` operation:

```
rs.status()
```

Step 12: Create additional users to address operational requirements. You can use *built-in roles* (page 85) to create common types of database users, such as the `dbOwner` (page 87) role to create a database administrator, the `readWrite` (page 86) role to create a user who can update data, or the `read` (page 85) role to create user who can search data but no more. You also can define *custom roles* (page 15).

For example, the following creates a database administrator for the `products` database:

```
use products
db.createUser(
  {
    user: "productsDBAdmin",
    pwd: "password",
    roles:
    [
      {
        role: "dbOwner",
        db: "products"
      }
    ]
  }
)
```

For an overview of roles and privileges, see *Authorization* (page 14). For more information on adding users, see *Manage User and Roles* (page 69).

3.3 Authentication Tutorials

The following tutorials provide instructions for MongoDB"s authentication related features.

Enable Client Access Control **(page 40)** Describes the process for enabling authentication for MongoDB deployments.

Enable Authentication in a Sharded Cluster **(page 41)** Control access to a sharded cluster through a key file and the `keyFile` setting on each of the cluster's components.

Enable Authentication after Creating the User Administrator **(page 42)** Describes an alternative process for enabling authentication for MongoDB deployments.

Use x.509 Certificates to Authenticate Clients **(page 44)** Use x.509 for client authentication.

Use x.509 Certificate for Membership Authentication **(page 46)** Use x.509 for internal member authentication for replica sets and sharded clusters.

Authenticate Using SASL and LDAP with ActiveDirectory **(page 49)** Describes the process for authentication using SASL/LDAP with ActiveDirectory.

Authenticate Using SASL and LDAP with OpenLDAP **(page 52)** Describes the process for authentication using SASL/LDAP with OpenLDAP.

Configure MongoDB with Kerberos Authentication on Linux **(page 55)** For MongoDB Enterprise Linux, describes the process to enable Kerberos-based authentication for MongoDB deployments.

Configure MongoDB with Kerberos Authentication on Windows **(page 59)** For MongoDB Enterprise for Windows, describes the process to enable Kerberos-based authentication for MongoDB deployments.

Authenticate to a MongoDB Instance or Cluster **(page 61)** Describes the process for authenticating to MongoDB systems using the `mongo` shell.

Generate a Key File **(page 62)** Use key file to allow the components of MongoDB sharded cluster or replica set to mutually authenticate.

Troubleshoot Kerberos Authentication on Linux **(page 63)** Steps to troubleshoot Kerberos-based authentication for MongoDB deployments.

Implement Field Level Redaction **(page 64)** Describes the process to set up and access document content that can have different access levels for the same data.

Enable Client Access Control

Overview

Enabling access control on a MongoDB instance restricts access to the instance by requiring that users identify themselves when connecting. In this procedure, you enable access control and then create the instance's first user, which must be a user administrator. The user administrator grants further access to the instance by creating additional users.

Considerations

If you create the user administrator before enabling access control, MongoDB disables the *localhost exception* (page 9). In that case, you must use the "*Enable Authentication after Creating the User Administrator* (page 42)" procedure to enable access control.

This procedure uses the *localhost exception* (page 9) to allow you to create the first user after enabling authentication. See *Localhost Exception* (page 9) and *Authentication* (page 6) for more information.

Procedure

Step 1: Start the MongoDB instance with authentication enabled. Start the `mongod` or `mongos` instance with the `authorization` or `keyFile` setting. Use `authorization` on a standalone instance. Use `keyFile` on an instance in a *replica set* or *sharded cluster*.

For example, to start a `mongod` with authentication enabled and a key file stored in `/private/var`, first set the following option in the `mongod`'s configuration file:

```
security:
  keyFile: /private/var/key.pem
```

Then start the `mongod` and specify the config file. For example:

```
mongod --config /etc/mongodb/mongodb.conf
```

After you enable authentication, only the user administrator can connect to the MongoDB instance. The user administrator must log in and grant further access to the instance by creating additional users.

Step 2: Connect to the MongoDB instance via the localhost exception. Connect to the MongoDB instance from a client running on the same system. This access is made possible by the *localhost exception* (page 9).

Step 3: Create the system user administrator. Add the user with the `userAdminAnyDatabase` (page 92) role, and only that role.

The following example creates the user `siteUserAdmin` user on the `admin` database:

```
use admin
db.createUser(
  {
    user: "siteUserAdmin",
    pwd: "password",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

After you create the user administrator, the *localhost exception* (page 9) is no longer available.

The `mongo` shell executes a number of commands at start up. As a result, when you log in as the user administrator, you may see authentication errors from one or more commands. You may ignore these errors, which are expected, because the `userAdminAnyDatabase` (page 92) role does not have permissions to run some of the start up commands.

Step 4: Create additional users. Login in with the user administrator's credentials and create additional users. See *Manage User and Roles* (page 69).

Next Steps

If you need to disable access control for any reason, restart the process without the `authorization` or `keyFile` setting.

Enable Authentication in a Sharded Cluster

New in version 2.0: Support for authentication with sharded clusters.

Overview

When authentication is enabled on a sharded cluster, every client that accesses the cluster must provide credentials. This includes MongoDB instances that access each other within the cluster.

To enable authentication on a sharded cluster, you must enable authentication individually on each component of the cluster. This means enabling authentication on each `mongos` and each `mongod`, including each config server, and all members of a shard's replica set.

Authentication requires an authentication mechanism and, in most cases, a `keyfile`. The content of the key file must be the same on all cluster members.

Considerations

It is not possible to convert an existing sharded cluster that does not enforce access control to require authentication without taking all components of the cluster offline for a short period of time.

As described in *Localhost Exception* (page 9), the localhost exception will apply to the individual shards unless you either create an administrative user or disable the localhost exception on each shard.

Procedure

Step 1: Create a key file. Create the key file your deployment will use to authenticate servers to each other.

To generate pseudo-random data to use for a `keyfile`, issue the following `openssl` command:

```
openssl rand -base64 741 > mongodb-keyfile
chmod 600 mongodb-keyfile
```

You may generate a key file using any method you choose. Always ensure that the password stored in the key file is both long and contains a high amount of entropy. Using `openssl` in this manner helps generate such a key.

Step 2: Enable authentication on each component in the cluster. On each `mongos` and `mongod` in the cluster, including all config servers and shards, specify the key file using one of the following approaches:

Specify the key file in the configuration file. In the configuration file, set the `keyFile` option to the key file's path and then start the component, as in the following example:

```
security:
  keyFile: /srv/mongodb/keyfile
```

Specify the key file at runtime. When starting the component, set the *`--keyFile`* option, which is an option for both `mongos` instances and `mongod` instances. Set the *`--keyFile`* to the key file's path. The `keyFile` setting implies the `authorization` setting, which means in most cases you do not need to set `authorization` explicitly.

Step 3: Add users. While connected to a `mongos`, add the first administrative user and then add subsequent users. See *Create a User Administrator* (page 67).

Related Documents

- *Authentication* (page 6)
- *Security* (page 3)
- *Use x.509 Certificate for Membership Authentication* (page 46)

Enable Authentication after Creating the User Administrator

Overview

Enabling authentication on a MongoDB instance restricts access to the instance by requiring that users identify themselves when connecting. In this procedure, you will create the instance's first user, which must be a user administrator and then enable authentication. Then, you can authenticate as the user administrator to create additional users and grant additional access to the instance.

This procedures outlines how enable authentication after creating the user administrator. The approach requires a restart. To enable authentication without restarting, see *Enable Client Access Control* (page 40).

Considerations

This document outlines a procedure for enabling authentication for MongoDB instance where you create the first user on an existing MongoDB system that does not require authentication before restarting the instance and requiring authentication. You can use the *localhost exception* (page 9) to gain access to a system with no users and authentication enabled. See *Enable Client Access Control* (page 40) for the description of that procedure.

Procedure

Step 1: Start the MongoDB instance without authentication. Start the `mongod` or `mongos` instance *without* the `authorization` or `keyFile` setting. For example:

```
mongod --port 27017 --dbpath /data/db1
```

For details on starting a `mongod` or `mongos`, see `http://docs.mongodb.org/manual/tutorial/manage-mongodb-pro` or `http://docs.mongodb.org/manual/tutorial/deploy-shard-cluster`.

Step 2: Create the system user administrator. Add the user with the `userAdminAnyDatabase` (page 92) role, and only that role.

The following example creates the user `siteUserAdmin` user on the `admin` database:

```
use admin
db.createUser(
  {
    user: "siteUserAdmin",
    pwd: "password",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

Step 3: Re-start the MongoDB instance with authentication enabled. Re-start the `mongod` or `mongos` instance with the `authorization` or `keyFile` setting. Use `authorization` on a standalone instance. Use `keyFile` on an instance in a *replica set* or *sharded cluster*.

The following example enables authentication on a standalone `mongod` using the `authorization` command-line option:

```
mongod --auth --config /etc/mongodb/mongodb.conf
```

Step 4: Create additional users. Log in with the user administrator's credentials and create additional users. See *Manage User and Roles* (page 69).

Next Steps

If you need to disable authentication for any reason, restart the process without the `authorization` or `keyFile` option.

Use x.509 Certificates to Authenticate Clients

New in version 2.6.

MongoDB supports x.509 certificate authentication for use with a secure *TLS/SSL connection* (page 26). The x.509 client authentication allows *clients to authenticate to servers with certificates* (page 44) rather than with a username and password.

To use x.509 authentication for the internal authentication of replica set/sharded cluster members, see *Use x.509 Certificate for Membership Authentication* (page 46).

Prerequisites

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, in particular x.509 certificates, and Certificate Authority is beyond the scope of this document. This tutorial assumes prior knowledge of TLS/SSL as well as access to valid x.509 certificates.

Certificate Authority For production use, your MongoDB deployment should use valid certificates generated and signed by a single certificate authority. You or your organization can generate and maintain an independent certificate authority, or use certificates generated by a third-party SSL vendor. Obtaining and managing certificates is beyond the scope of this documentation.

Client x.509 Certificate The client certificate must have the following properties:

- A single Certificate Authority (CA) must issue the certificates for both the client and the server.
- Client certificates must contain the following fields:

  ```
  keyUsage = digitalSignature
  extendedKeyUsage = clientAuth
  ```

- Each unique MongoDB user must have a unique certificate.
- A client x.509 certificate's subject, which contains the Distinguished Name (`DN`), must **differ** from that of a *Member x.509 Certificate* (page 46). Specifically, the subjects must differ with regards to at least one of the following attributes: Organization (`O`), the Organizational Unit (`OU`) or the Domain Component (`DC`).

  Warning: If a client x.509 certificate's subject has the same `O`, `OU`, and `DC` combination as the *Member x.509 Certificate* (page 46), the client will be identified as a cluster member and granted full permission on the system.

Procedures

Configure MongoDB Server

Use Command-line Options You can configure the MongoDB server from the command line, e.g.:

```
mongod --clusterAuthMode x509 --sslMode requireSSL --sslPEMKeyFile <path to SSL certificate and key f
```

Warning: If the `--sslCAFile` option and its target file are not specified, x.509 client and member authentication will not function. `mongod`, and `mongos` in sharded systems, will not be able to verify the certificates of processes connecting to it against the trusted certificate authority (CA) that issued them, breaking the certificate chain.
As of version 2.6.4, `mongod` will not start with x.509 authentication enabled if the CA file is not specified.

Use Configuration File You may also specify these options in the `configuration file`.

Starting in MongoDB 2.6, you can specify the configuration for MongoDB in `YAML format`, e.g.:

```
security:
   clusterAuthMode: x509
net:
   ssl:
      mode: requireSSL
      PEMKeyFile: <path to TLS/SSL certificate and key PEM file>
      CAFile: <path to root CA PEM file>
```

For backwards compatibility, you can also specify the configuration using the older configuration file format[60], e.g.:

```
clusterAuthMode = x509
sslMode = requireSSL
sslPEMKeyFile = <path to TLS/SSL certificate and key PEM file>
sslCAFile = <path to the root CA PEM file>
```

Include any additional options, TLS/SSL or otherwise, that are required for your specific configuration.

Add x.509 Certificate `subject` as a User To authenticate with a client certificate, you must first add the value of the `subject` from the client certificate as a MongoDB user. Each unique x.509 client certificate corresponds to a single MongoDB user; i.e. you cannot use a single client certificate to authenticate more than one MongoDB user.

1. You can retrieve the `subject` from the client certificate with the following command:

   ```
   openssl x509 -in <pathToClient PEM> -inform PEM -subject -nameopt RFC2253
   ```

   The command returns the `subject` string as well as certificate:

   ```
   subject= CN=myName,OU=myOrgUnit,O=myOrg,L=myLocality,ST=myState,C=myCountry
   -----BEGIN CERTIFICATE-----
   # ...
   -----END CERTIFICATE-----
   ```

2. Add the value of the `subject`, omitting the spaces, from the certificate as a user.

   For example, in the `mongo` shell, to add the user with both the `readWrite` role in the `test` database and the `userAdminAnyDatabase` role which is defined only in the `admin` database:

   ```
   db.getSiblingDB("$external").runCommand(
     {
       createUser: "CN=myName,OU=myOrgUnit,O=myOrg,L=myLocality,ST=myState,C=myCountry",
       roles: [
                { role: 'readWrite', db: 'test' },
                { role: 'userAdminAnyDatabase', db: 'admin' }
              ],
       writeConcern: { w: "majority" , wtimeout: 5000 }
   ```

[60] http://docs.mongodb.org/v2.4/reference/configuration-options

```
  }
)
```

In the above example, to add the user with the `readWrite` role in the `test` database, the role specification document specified `'test'` in the `db` field. To add `userAdminAnyDatabase` role for the user, the above example specified `'admin'` in the `db` field.

Note: Some roles are defined only in the `admin` database, including: `clusterAdmin`, `readAnyDatabase`, `readWriteAnyDatabase`, `dbAdminAnyDatabase`, and `userAdminAnyDatabase`. To add a user with these roles, specify `'admin'` in the `db`.

See *Manage User and Roles* (page 69) for details on adding a user with roles.

Authenticate with a x.509 Certificate To authenticate with a client certificate, you must first add a MongoDB user that corresponds to the client certificate. See *Add x.509 Certificate subject as a User* (page 45).

To authenticate, use the `db.auth()` method in the `$external` database, specifying `"MONGODB-X509"` for the `mechanism` field, and the *user that corresponds to the client certificate* (page 45) for the `user` field.

For example, if using the `mongo` shell,

1. Connect `mongo` shell to the `mongod` set up for SSL:

   ```
   mongo --ssl --sslPEMKeyFile <path to CA signed client PEM file> --sslCAFile <path to root CA PEM
   ```

2. To perform the authentication, use the `db.auth()` method in the `$external` database. For the `mechanism` field, specify `"MONGODB-X509"`, and for the `user` field, specify the user, or the `subject`, that corresponds to the client certificate.

   ```
   db.getSiblingDB("$external").auth(
     {
       mechanism: "MONGODB-X509",
       user: "CN=myName,OU=myOrgUnit,O=myOrg,L=myLocality,ST=myState,C=myCountry"
     }
   )
   ```

Use x.509 Certificate for Membership Authentication

New in version 2.6.

MongoDB supports x.509 certificate authentication for use with a secure *TLS/SSL connection* (page 26). Sharded cluster members and replica set members can use x.509 certificates to verify their membership to the cluster or the replica set instead of using *keyfiles* (page 6). The membership authentication is an internal process.

For client authentication with x.509, see *Use x.509 Certificates to Authenticate Clients* (page 44).

Important: A full description of TLS/SSL, PKI (Public Key Infrastructure) certificates, in particular x.509 certificates, and Certificate Authority is beyond the scope of this document. This tutorial assumes prior knowledge of TLS/SSL as well as access to valid x.509 certificates.

Member x.509 Certificate

The member certificate, used for internal authentication to verify membership to the sharded cluster or a replica set, must have the following properties:

- A single Certificate Authority (CA) must issue all the x.509 certificates for the members of a sharded cluster or a replica set.
- The Distinguished Name (DN), found in the member certificate's subject, must specify a non-empty value for *at least one* of the following attributes: Organization (O), the Organizational Unit (OU) or the Domain Component (DC).
- The Organization attributes (O's), the Organizational Unit attributes (OU's), and the Domain Components (DC's) must match those from the certificates for the other cluster members. To match, the certificate must match all specifications of these attributes, or even the non-specification of these attributes. The order of the attributes does not matter.

  In the following example, the two DN's contain matching specifications for O, OU as well as the non-specification of the DC attribute.

  ```
  CN=host1,OU=Dept1,O=MongoDB,ST=NY,C=US
  C=US, ST=CA, O=MongoDB, OU=Dept1, CN=host2
  ```

  However, the following two DN's contain a mismatch for the OU attribute since one contains two OU specifications and the other, only one specification.

  ```
  CN=host1,OU=Dept1,OU=Sales,O=MongoDB
  CN=host2,OU=Dept1,O=MongoDB
  ```

- Either the Common Name (CN) or one of the Subject Alternative Name (SAN) entries must match the hostname of the server, used by the other members of the cluster.

  For example, the certificates for a cluster could have the following subjects:

  ```
  subject= CN=<myhostname1>,OU=Dept1,O=MongoDB,ST=NY,C=US
  subject= CN=<myhostname2>,OU=Dept1,O=MongoDB,ST=NY,C=US
  subject= CN=<myhostname3>,OU=Dept1,O=MongoDB,ST=NY,C=US
  ```

You *can* use an x509 certificate that does not have Extended Key Usage (EKU) attributes set. If you use EKU attribute in the PEMKeyFile certificate, then specify the clientAuth and/or serverAuth attributes (i.e. "TLS Web Client Authentication" and "TLS Web Server Authentication,") as needed. The certificate that you specify for the PEMKeyFile option requires the serverAuth attribute, and the certificate you specify to clusterFile requires the clientAuth attribute. If you omit ClusterFile, mongod will use the certificate specified to PEMKeyFile for member authentication.

Configure Replica Set/Sharded Cluster

Use Command-line Options To specify the x.509 certificate for internal cluster member authentication, append the additional TLS/SSL options *--clusterAuthMode* and *--sslClusterFile*, as in the following example for a member of a replica set:

```
mongod --replSet <name> --sslMode requireSSL --clusterAuthMode x509 --sslClusterFile <path to members
```

Include any additional options, TLS/SSL or otherwise, that are required for your specific configuration. For instance, if the membership key is encrypted, set the *--sslClusterPassword* to the passphrase to decrypt the key or have MongoDB prompt for the passphrase. See *SSL Certificate Passphrase* (page 30) for details.

Warning: If the *--sslCAFile* option and its target file are not specified, x.509 client and member authentication will not function. mongod, and mongos in sharded systems, will not be able to verify the certificates of processes connecting to it against the trusted certificate authority (CA) that issued them, breaking the certificate chain.
As of version 2.6.4, mongod will not start with x.509 authentication enabled if the CA file is not specified.

Use Configuration File You can specify the configuration for MongoDB in a `YAML formatted` configuration file, as in the following example:

```
security:
   clusterAuthMode: x509
net:
   ssl:
      mode: requireSSL
      PEMKeyFile: <path to TLS/SSL certificate and key PEM file>
      CAFile: <path to root CA PEM file>
      clusterFile: <path to x.509 membership certificate and key PEM file>
```

See `security.clusterAuthMode`, `net.ssl.mode`, `net.ssl.PEMKeyFile`, `net.ssl.CAFile`, and `net.ssl.clusterFile` for more information on the settings.

Upgrade from Keyfile Authentication to x.509 Authentication

To upgrade clusters that are currently using keyfile authentication to x.509 authentication, use a rolling upgrade process.

Clusters Currently Using TLS/SSL For clusters using TLS/SSL and keyfile authentication, to upgrade to x.509 cluster authentication, use the following rolling upgrade process:

1. For each node of a cluster, start the node with the option *`--clusterAuthMode`* set to `sendKeyFile` and the option *`--sslClusterFile`* set to the appropriate path of the node's certificate. Include other *TLS/SSL options* (page 26) as well as any other options that are required for your specific configuration. For example:

   ```
   mongod --replSet <name> --sslMode requireSSL --clusterAuthMode sendKeyFile --sslClusterFile <pat
   ```

   With this setting, each node continues to use its keyfile to authenticate itself as a member. However, each node can now accept either a keyfile or an x.509 certificate from other members to authenticate those members. Upgrade all nodes of the cluster to this setting.

2. Then, for each node of a cluster, connect to the node and use the `setParameter` command to update the `clusterAuthMode` to `sendX509`. [61] For example,

   ```
   db.getSiblingDB('admin').runCommand( { setParameter: 1, clusterAuthMode: "sendX509" } )
   ```

   With this setting, each node uses its x.509 certificate, specified with the *`--sslClusterFile`* option in the previous step, to authenticate itself as a member. However, each node continues to accept either a keyfile or an x.509 certificate from other members to authenticate those members. Upgrade all nodes of the cluster to this setting.

3. Optional but recommended. Finally, for each node of the cluster, connect to the node and use the `setParameter` command to update the `clusterAuthMode` to `x509` to only use the x.509 certificate for authentication. [1] For example:

   ```
   db.getSiblingDB('admin').runCommand( { setParameter: 1, clusterAuthMode: "x509" } )
   ```

4. After the upgrade of all nodes, edit the `configuration file` with the appropriate x.509 settings to ensure that upon subsequent restarts, the cluster uses x.509 authentication.

See *`--clusterAuthMode`* for the various modes and their descriptions.

[61] As an alternative to using the `setParameter` command, you can also restart the nodes with the appropriate TLS/SSL and x509 options and values.

Clusters Currently Not Using TLS/SSL For clusters using keyfile authentication but not TLS/SSL, to upgrade to x.509 authentication, use the following rolling upgrade process:

1. For each node of a cluster, start the node with the option `--sslMode` set to `allowSSL`, the option `--clusterAuthMode` set to `sendKeyFile` and the option `--sslClusterFile` set to the appropriate path of the node's certificate. Include other *TLS/SSL options* (page 26) as well as any other options that are required for your specific configuration. For example:

   ```
   mongod --replSet <name> --sslMode allowSSL --clusterAuthMode sendKeyFile --sslClusterFile <path
   ```

   The `--sslMode allowSSL` setting allows the node to accept both TLS/SSL and non-TLS/non-SSL incoming connections. Its outgoing connections do not use TLS/SSL.

   The `--clusterAuthMode sendKeyFile` setting allows each node continues to use its keyfile to authenticate itself as a member. However, each node can now accept either a keyfile or an x.509 certificate from other members to authenticate those members.

   Upgrade all nodes of the cluster to these settings.

2. Then, for each node of a cluster, connect to the node and use the `setParameter` command to update the `sslMode` to `preferSSL` and the `clusterAuthMode` to `sendX509`. [1] For example:

   ```
   db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "preferSSL", clusterAuthMode: "
   ```

   With the `sslMode` set to `preferSSL`, the node accepts both TLS/SSL and non-TLS/non-SSL incoming connections, and its outgoing connections use TLS/SSL.

   With the `clusterAuthMode` set to `sendX509`, each node uses its x.509 certificate, specified with the `--sslClusterFile` option in the previous step, to authenticate itself as a member. However, each node continues to accept either a keyfile or an x.509 certificate from other members to authenticate those members.

   Upgrade all nodes of the cluster to these settings.

3. Optional but recommended. Finally, for each node of the cluster, connect to the node and use the `setParameter` command to update the `sslMode` to `requireSSL` and the `clusterAuthMode` to `x509`. [1] For example:

   ```
   db.getSiblingDB('admin').runCommand( { setParameter: 1, sslMode: "requireSSL", clusterAuthMode:
   ```

   With the `sslMode` set to `requireSSL`, the node only uses TLS/SSLs connections.

   With the `clusterAuthMode` set to `x509`, the node only uses the x.509 certificate for authentication.

4. After the upgrade of all nodes, edit the `configuration file` with the appropriate TLS/SSL and x.509 settings to ensure that upon subsequent restarts, the cluster uses x.509 authentication.

See `--clusterAuthMode` for the various modes and their descriptions.

Authenticate Using SASL and LDAP with ActiveDirectory

MongoDB Enterprise provides support for proxy authentication of users. This allows administrators to configure a MongoDB cluster to authenticate users by proxying authentication requests to a specified Lightweight Directory Access Protocol (LDAP) service.

Considerations

MongoDB Enterprise for Windows does **not** include LDAP support for authentication. However, MongoDB Enterprise for Linux supports using LDAP authentication with an ActiveDirectory server.

MongoDB does **not** support LDAP authentication in mixed sharded cluster deployments that contain both version 2.4 and version 2.6 shards. See `http://docs.mongodb.org/manual/release-notes/2.6-upgrade` for upgrade instructions.

Use secure encrypted or trusted connections between clients and the server, as well as between `saslauthd` and the LDAP server. The LDAP server uses the `SASL PLAIN` mechanism, sending and receiving data in **plain text**. You should use only a trusted channel such as a VPN, a connection encrypted with TLS/SSL, or a trusted wired network.

Configure `saslauthd`

LDAP support for user authentication requires proper configuration of the `saslauthd` daemon process as well as the MongoDB server.

Step 1: Specify the mechanism. On systems that configure `saslauthd` with the `/etc/sysconfig/saslauthd` file, such as Red Hat Enterprise Linux, Fedora, CentOS, and Amazon Linux AMI, set the mechanism `MECH` to `ldap`:

```
MECH=ldap
```

On systems that configure `saslauthd` with the `/etc/default/saslauthd` file, such as Ubuntu, set the `MECHANISMS` option to `ldap`:

```
MECHANISMS="ldap"
```

Step 2: Adjust caching behavior. On certain Linux distributions, `saslauthd` starts with the caching of authentication credentials *enabled*. Until restarted or until the cache expires, `saslauthd` will not contact the LDAP server to re-authenticate users in its authentication cache. This allows `saslauthd` to successfully authenticate users in its cache, even in the LDAP server is down or if the cached users' credentials are revoked.

To set the expiration time (in seconds) for the authentication cache, see the -t option[62] of `saslauthd`.

Step 3: Configure LDAP Options with ActiveDirectory. If the `saslauthd.conf` file does not exist, create it. The `saslauthd.conf` file usually resides in the `/etc` folder. If specifying a different file path, see the -O option[63] of `saslauthd`.

To use with ActiveDirectory, start `saslauthd` with the following configuration options set in the `saslauthd.conf` file:

```
ldap_servers: <ldap uri>
ldap_use_sasl: yes
ldap_mech: DIGEST-MD5
ldap_auth_method: fastbind
```

For the `<ldap uri>`, specify the uri of the ldap server. For example, `ldap_servers: ldaps://ad.example.net`.

For more information on `saslauthd` configuration, see http://www.openldap.org/doc/admin24/guide.html#Configuringsaslauthd.

Step 4: Test the `saslauthd` configuration. Use `testsaslauthd` utility to test the `saslauthd` configuration. For example:

[62] http://www.linuxcommand.org/man_pages/saslauthd8.html
[63] http://www.linuxcommand.org/man_pages/saslauthd8.html

```
testsaslauthd -u testuser -p testpassword -f /var/run/saslauthd/mux
```

Note: `/var/run/saslauthd` directory must have permissions set to `755` for MongoDB to successfully authenticate.

Configure MongoDB

Step 1: Add user to MongoDB for authentication. Add the user to the `$external` database in MongoDB. To specify the user's privileges, assign *roles* (page 14) to the user.

For example, the following adds a user with read-only access to the `records` database.

```
db.getSiblingDB("$external").createUser(
    {
      user : <username>,
      roles: [ { role: "read", db: "records" } ]
    }
)
```

Add additional principals as needed. For more information about creating and managing users, see `http://docs.mongodb.org/manual/reference/command/nav-user-management`.

Step 2: Configure MongoDB server. To configure the MongoDB server to use the `saslauthd` instance for proxy authentication, start the `mongod` with the following options:

- `--auth`,
- `authenticationMechanisms` parameter set to `PLAIN`, and
- `saslauthdPath` parameter set to the path to the Unix-domain Socket of the `saslauthd` instance.

Configure the MongoDB server using either the command line option `--setParameter` or the `configuration file`. Specify additional configurations as appropriate for your configuration.

If you use the `authorization` option to enforce authentication, you will need privileges to create a user.

Use specific `saslauthd` socket path. For socket path of `/<some>/<path>/saslauthd`, set the `saslauthdPath` to `/<some>/<path>/saslauthd/mux`, as in the following command line example:

```
mongod --auth --setParameter saslauthdPath=/<some>/<path>/saslauthd/mux --setParameter authentication
```

Or if using a `YAML format configuration file`, specify the following settings in the file:

```
security:
   authorization: enabled

setParameter:
   saslauthdPath: /<some>/<path>/saslauthd/mux
   authenticationMechanisms: PLAIN
```

Or, if using the older configuration file format[64]:

```
auth=true
setParameter=saslauthdPath=/<some>/<path>/saslauthd/mux
setParameter=authenticationMechanisms=PLAIN
```

[64] http://docs.mongodb.org/v2.4/reference/configuration-options

Use default Unix-domain socket path. To use the default Unix-domain socket path, set the `saslauthdPath` to the empty string `""`, as in the following command line example:

```
mongod --auth --setParameter saslauthdPath="" --setParameter authenticationMechanisms=PLAIN
```

Or if using a `YAML format configuration file`, specify the following settings in the file:

```
security:
   authorization: enabled

setParameter:
   saslauthdPath: ""
   authenticationMechanisms: PLAIN
```

Or, if using the older configuration file format[65]:

```
auth=true
setParameter=saslauthdPath=""
setParameter=authenticationMechanisms=PLAIN
```

Step 3: Authenticate the user in the `mongo` shell. To perform the authentication in the `mongo` shell, use the `db.auth()` method in the `$external` database.

Specify the value `"PLAIN"` in the `mechanism` field, the user and password in the `user` and `pwd` fields respectively, and the value `false` in the `digestPassword` field. You **must** specify `false` for `digestPassword` since the server must receive an undigested password to forward on to `saslauthd`, as in the following example:

```
db.getSiblingDB("$external").auth(
   {
     mechanism: "PLAIN",
     user: <username>,
     pwd:  <cleartext password>,
     digestPassword: false
   }
)
```

The server forwards the password in plain text. In general, use only on a trusted channel (VPN, TLS/SSL, trusted wired network). See Considerations.

Authenticate Using SASL and LDAP with OpenLDAP

MongoDB Enterprise provides support for proxy authentication of users. This allows administrators to configure a MongoDB cluster to authenticate users by proxying authentication requests to a specified Lightweight Directory Access Protocol (LDAP) service.

Considerations

MongoDB Enterprise for Windows does **not** include LDAP support for authentication. However, MongoDB Enterprise for Linux supports using LDAP authentication with an ActiveDirectory server.

MongoDB does **not** support LDAP authentication in mixed sharded cluster deployments that contain both version 2.4 and version 2.6 shards. See `http://docs.mongodb.org/manual/release-notes/2.6-upgrade` for upgrade instructions.

[65] http://docs.mongodb.org/v2.4/reference/configuration-options

Use secure encrypted or trusted connections between clients and the server, as well as between `saslauthd` and the LDAP server. The LDAP server uses the `SASL PLAIN` mechanism, sending and receiving data in **plain text**. You should use only a trusted channel such as a VPN, a connection encrypted with TLS/SSL, or a trusted wired network.

Configure `saslauthd`

LDAP support for user authentication requires proper configuration of the `saslauthd` daemon process as well as the MongoDB server.

Step 1: Specify the mechanism. On systems that configure `saslauthd` with the `/etc/sysconfig/saslauthd` file, such as Red Hat Enterprise Linux, Fedora, CentOS, and Amazon Linux AMI, set the mechanism `MECH` to `ldap`:

```
MECH=ldap
```

On systems that configure `saslauthd` with the `/etc/default/saslauthd` file, such as Ubuntu, set the `MECHANISMS` option to `ldap`:

```
MECHANISMS="ldap"
```

Step 2: Adjust caching behavior. On certain Linux distributions, `saslauthd` starts with the caching of authentication credentials *enabled*. Until restarted or until the cache expires, `saslauthd` will not contact the LDAP server to re-authenticate users in its authentication cache. This allows `saslauthd` to successfully authenticate users in its cache, even in the LDAP server is down or if the cached users' credentials are revoked.

To set the expiration time (in seconds) for the authentication cache, see the -t option[66] of `saslauthd`.

Step 3: Configure LDAP Options with OpenLDAP. If the `saslauthd.conf` file does not exist, create it. The `saslauthd.conf` file usually resides in the `/etc` folder. If specifying a different file path, see the -O option[67] of `saslauthd`.

To connect to an OpenLDAP server, update the `saslauthd.conf` file with the following configuration options:

```
ldap_servers: <ldap uri>
ldap_search_base: <search base>
ldap_filter: <filter>
```

The `ldap_servers` specifies the uri of the LDAP server used for authentication. In general, for OpenLDAP installed on the local machine, you can specify the value `ldap://localhost:389` or if using LDAP over TLS/SSL, you can specify the value `ldaps://localhost:636`.

The `ldap_search_base` specifies distinguished name to which the search is relative. The search includes the base or objects below.

The `ldap_filter` specifies the search filter.

The values for these configuration options should correspond to the values specific for your test. For example, to filter on email, specify `ldap_filter: (mail=%n)` instead.

OpenLDAP Example A sample `saslauthd.conf` file for OpenLDAP includes the following content:

[66] http://www.linuxcommand.org/man_pages/saslauthd8.html
[67] http://www.linuxcommand.org/man_pages/saslauthd8.html

```
ldap_servers: ldaps://ad.example.net
ldap_search_base: ou=Users,dc=example,dc=com
ldap_filter: (uid=%u)
```

To use this sample OpenLDAP configuration, create users with a `uid` attribute (login name) and place under the `Users` organizational unit (`ou`) under the domain components (`dc`) `example` and `com`.

For more information on `saslauthd` configuration, see http://www.openldap.org/doc/admin24/guide.html#Configuringsaslauthd.

Step 4: Test the `saslauthd` configuration. Use `testsaslauthd` utility to test the `saslauthd` configuration. For example:

```
testsaslauthd -u testuser -p testpassword -f /var/run/saslauthd/mux
```

Note: `/var/run/saslauthd` directory must have permissions set to `755` for MongoDB to successfully authenticate.

Configure MongoDB

Step 1: Add user to MongoDB for authentication. Add the user to the `$external` database in MongoDB. To specify the user's privileges, assign *roles* (page 14) to the user.

For example, the following adds a user with read-only access to the `records` database.

```
db.getSiblingDB("$external").createUser(
    {
      user : <username>,
      roles: [ { role: "read", db: "records" } ]
    }
)
```

Add additional principals as needed. For more information about creating and managing users, see `http://docs.mongodb.org/manual/reference/command/nav-user-management`.

Step 2: Configure MongoDB server. To configure the MongoDB server to use the `saslauthd` instance for proxy authentication, start the `mongod` with the following options:

- *`--auth`*,
- `authenticationMechanisms` parameter set to `PLAIN`, and
- `saslauthdPath` parameter set to the path to the Unix-domain Socket of the `saslauthd` instance.

Configure the MongoDB server using either the command line option `--setParameter` or the `configuration file`. Specify additional configurations as appropriate for your configuration.

If you use the `authorization` option to enforce authentication, you will need privileges to create a user.

Use specific `saslauthd` socket path. For socket path of `/<some>/<path>/saslauthd`, set the `saslauthdPath` to `/<some>/<path>/saslauthd/mux`, as in the following command line example:

```
mongod --auth --setParameter saslauthdPath=/<some>/<path>/saslauthd/mux --setParameter authentication
```

Or if using a `YAML format configuration file`, specify the following settings in the file:

```
security:
   authorization: enabled

setParameter:
   saslauthdPath: /<some>/<path>/saslauthd/mux
   authenticationMechanisms: PLAIN
```

Or, if using the older configuration file format[68]:

```
auth=true
setParameter=saslauthdPath=/<some>/<path>/saslauthd/mux
setParameter=authenticationMechanisms=PLAIN
```

Use default Unix-domain socket path. To use the default Unix-domain socket path, set the `saslauthdPath` to the empty string `""`, as in the following command line example:

```
mongod --auth --setParameter saslauthdPath="" --setParameter authenticationMechanisms=PLAIN
```

Or if using a `YAML format configuration file`, specify the following settings in the file:

```
security:
   authorization: enabled

setParameter:
   saslauthdPath: ""
   authenticationMechanisms: PLAIN
```

Or, if using the older configuration file format[69]:

```
auth=true
setParameter=saslauthdPath=""
setParameter=authenticationMechanisms=PLAIN
```

Step 3: Authenticate the user in the `mongo` shell. To perform the authentication in the `mongo` shell, use the `db.auth()` method in the `$external` database.

Specify the value `"PLAIN"` in the `mechanism` field, the user and password in the `user` and `pwd` fields respectively, and the value `false` in the `digestPassword` field. You **must** specify `false` for `digestPassword` since the server must receive an undigested password to forward on to `saslauthd`, as in the following example:

```
db.getSiblingDB("$external").auth(
   {
     mechanism: "PLAIN",
     user: <username>,
     pwd:  <cleartext password>,
     digestPassword: false
   }
)
```

The server forwards the password in plain text. In general, use only on a trusted channel (VPN, TLS/SSL, trusted wired network). See Considerations.

Configure MongoDB with Kerberos Authentication on Linux

New in version 2.4.

[68] http://docs.mongodb.org/v2.4/reference/configuration-options
[69] http://docs.mongodb.org/v2.4/reference/configuration-options

Overview

MongoDB Enterprise supports authentication using a *Kerberos service* (page 11). Kerberos is an industry standard authentication protocol for large client/server system.

Prerequisites

Setting up and configuring a Kerberos deployment is beyond the scope of this document. This tutorial assumes you have configured a *Kerberos service principal* (page 12) for each `mongod` and `mongos` instance in your MongoDB deployment, and you have a valid *keytab file* (page 12) for for each `mongod` and `mongos` instance.

To verify MongoDB Enterprise binaries:

```
mongod --version
```

In the output from this command, look for the string `modules: subscription` or `modules: enterprise` to confirm your system has MongoDB Enterprise.

Procedure

The following procedure outlines the steps to add a Kerberos user principal to MongoDB, configure a standalone `mongod` instance for Kerberos support, and connect using the `mongo` shell and authenticate the user principal.

Step 1: Start `mongod` without Kerberos. For the initial addition of Kerberos users, start `mongod` without Kerberos support.

If a Kerberos user is already in MongoDB and has the *privileges required to create a user*, you can start `mongod` with Kerberos support.

Step 2: Connect to `mongod`. Connect via the `mongo` shell to the `mongod` instance. If `mongod` has *`--auth`* enabled, ensure you connect with the *privileges required to create a user*.

Step 3: Add Kerberos Principal(s) to MongoDB. Add a Kerberos principal, `<username>@<KERBEROS REALM>` or `<username>/<instance>@<KERBEROS REALM>`, to MongoDB in the `$external` database. Specify the Kerberos realm in all uppercase. The `$external` database allows `mongod` to consult an external source (e.g. Kerberos) to authenticate. To specify the user's privileges, assign *roles* (page 14) to the user.

The following example adds the Kerberos principal `application/reporting@EXAMPLE.NET` with read-only access to the `records` database:

```
use $external
db.createUser(
   {
     user: "application/reporting@EXAMPLE.NET",
     roles: [ { role: "read", db: "records" } ]
   }
)
```

Add additional principals as needed. For every user you want to authenticate using Kerberos, you must create a corresponding user in MongoDB. For more information about creating and managing users, see `http://docs.mongodb.org/manual/reference/command/nav-user-management`.

Step 4: Start `mongod` with Kerberos support. To start `mongod` with Kerberos support, set the environmental variable `KRB5_KTNAME` to the path of the keytab file and the `mongod` parameter `authenticationMechanisms` to `GSSAPI` in the following form:

```
env KRB5_KTNAME=<path to keytab file> \
mongod \
--setParameter authenticationMechanisms=GSSAPI
<additional mongod options>
```

For example, the following starts a standalone `mongod` instance with Kerberos support:

```
env KRB5_KTNAME=/opt/mongodb/mongod.keytab \
/opt/mongodb/bin/mongod --auth \
--setParameter authenticationMechanisms=GSSAPI \
--dbpath /opt/mongodb/data
```

The path to your `mongod` as well as your *keytab file* (page 12) may differ. Modify or include additional `mongod` options as required for your configuration. The *keytab file* (page 12) must be only accessible to the owner of the `mongod` process.

With the official `.deb` or `.rpm` packages, you can set the `KRB5_KTNAME` in a environment settings file. See *KRB5_KTNAME* (page 57) for details.

Step 5: Connect `mongo` shell to `mongod` and authenticate. Connect the `mongo` shell client as the Kerberos principal `application/reporting@EXAMPLE.NET`. Before connecting, you must have used Kerberos's `kinit` program to get credentials for `application/reporting@EXAMPLE.NET`.

You can connect and authenticate from the command line.

```
mongo --authenticationMechanism=GSSAPI --authenticationDatabase='$external' \
--username application/reporting@EXAMPLE.NET
```

Or, alternatively, you can first connect `mongo` to the `mongod`, and then from the `mongo` shell, use the `db.auth()` method to authenticate in the `$external` database.

```
use $external
db.auth( { mechanism: "GSSAPI", user: "application/reporting@EXAMPLE.NET" } )
```

Additional Considerations

KRB5_KTNAME If you installed MongoDB Enterprise using one of the official `.deb` or `.rpm` packages, and you use the included init/upstart scripts to control the `mongod` instance, you can set the `KR5_KTNAME` variable in the default environment settings file instead of setting the variable each time.

For `.rpm` packages, the default environment settings file is `/etc/sysconfig/mongod`.

For `.deb` packages, the file is `/etc/default/mongodb`.

Set the `KRB5_KTNAME` value in a line that resembles the following:

```
export KRB5_KTNAME="<path to keytab>"
```

Configure `mongos` for Kerberos To start `mongos` with Kerberos support, set the environmental variable `KRB5_KTNAME` to the path of its *keytab file* (page 12) and the `mongos` parameter `authenticationMechanisms` to `GSSAPI` in the following form:

```
env KRB5_KTNAME=<path to keytab file> \
mongos \
--setParameter authenticationMechanisms=GSSAPI \
<additional mongos options>
```

For example, the following starts a `mongos` instance with Kerberos support:

```
env KRB5_KTNAME=/opt/mongodb/mongos.keytab \
mongos \
--setParameter authenticationMechanisms=GSSAPI \
--configdb shard0.example.net, shard1.example.net,shard2.example.net \
--keyFile /opt/mongodb/mongos.keyfile
```

The path to your `mongos` as well as your *keytab file* (page 12) may differ. The *keytab file* (page 12) must be only accessible to the owner of the `mongos` process.

Modify or include any additional `mongos` options as required for your configuration. For example, instead of using `--keyFile` for internal authentication of sharded cluster members, you can use *x.509 member authentication* (page 46) instead.

Use a Config File To configure `mongod` or `mongos` for Kerberos support using a `configuration file`, specify the `authenticationMechanisms` setting in the configuration file:

If using the `YAML configuration file format`:

```
setParameter:
   authenticationMechanisms: GSSAPI
```

Or, if using the older `.ini` configuration file format:

```
setParameter=authenticationMechanisms=GSSAPI
```

Modify or include any additional `mongod` options as required for your configuration. For example, if `/opt/mongodb/mongod.conf` contains the following configuration settings for a standalone `mongod`:

```
security:
   authorization: enabled
setParameter:
   authenticationMechanisms: GSSAPI
storage:
   dbPath: /opt/mongodb/data
```

Or, if using the older configuration file format[70]:

```
auth = true
setParameter=authenticationMechanisms=GSSAPI
dbpath=/opt/mongodb/data
```

To start `mongod` with Kerberos support, use the following form:

```
env KRB5_KTNAME=/opt/mongodb/mongod.keytab \
/opt/mongodb/bin/mongod --config /opt/mongodb/mongod.conf
```

The path to your `mongod`, *keytab file* (page 12), and configuration file may differ. The *keytab file* (page 12) must be only accessible to the owner of the `mongod` process.

Troubleshoot Kerberos Setup for MongoDB If you encounter problems when starting `mongod` or `mongos` with Kerberos authentication, see *Troubleshoot Kerberos Authentication on Linux* (page 63).

[70] http://docs.mongodb.org/v2.4/reference/configuration-options

Incorporate Additional Authentication Mechanisms Kerberos authentication (*GSSAPI* (page 8) (Kerberos)) can work alongside MongoDB's challenge/response authentication mechanisms (*SCRAM-SHA-1* (page 6) and *MONGODB-CR* (page 7)), MongoDB's authentication mechanism for LDAP (*PLAIN* (page 8) (LDAP SASL)), and MongoDB's authentication mechanism for x.509 (*MONGODB-X509* (page 8)). Specify the mechanisms as follows:

```
--setParameter authenticationMechanisms=GSSAPI,SCRAM-SHA-1
```

Only add the other mechanisms if in use. This parameter setting does not affect MongoDB's internal authentication of cluster members.

Additional Resources

- MongoDB LDAP and Kerberos Authentication with Dell (Quest) Authentication Services[71]
- MongoDB with Red Hat Enterprise Linux Identity Management and Kerberos[72]

Configure MongoDB with Kerberos Authentication on Windows

New in version 2.6.

Overview

MongoDB Enterprise supports authentication using a *Kerberos service* (page 11). Kerberos is an industry standard authentication protocol for large client/server system. Kerberos allows MongoDB and applications to take advantage of existing authentication infrastructure and processes.

Prerequisites

Setting up and configuring a Kerberos deployment is beyond the scope of this document. This tutorial assumes have configured a *Kerberos service principal* (page 12) for each `mongod.exe` and `mongos.exe` instance.

Procedures

Step 1: Start `mongod.exe` without Kerberos. For the initial addition of Kerberos users, start `mongod.exe` without Kerberos support.

If a Kerberos user is already in MongoDB and has the *privileges required to create a user*, you can start `mongod.exe` with Kerberos support.

Step 2: Connect to `mongod`. Connect via the `mongo.exe` shell to the `mongod.exe` instance. If `mongod.exe` has `--auth` enabled, ensure you connect with the *privileges required to create a user*.

Step 3: Add Kerberos Principal(s) to MongoDB. Add a Kerberos principal, `<username>@<KERBEROS REALM>`, to MongoDB in the `$external` database. Specify the Kerberos realm in **ALL UPPERCASE**. The `$external` database allows `mongod.exe` to consult an external source (e.g. Kerberos) to authenticate. To specify the user's privileges, assign *roles* (page 14) to the user.

[71] https://www.mongodb.com/blog/post/mongodb-ldap-and-kerberos-authentication-dell-quest-authentication-services?jmp=docs
[72] http://docs.mongodb.org/ecosystem/tutorial/manage-red-hat-enterprise-linux-identity-management?jmp=docs

The following example adds the Kerberos principal `reportingapp@EXAMPLE.NET` with read-only access to the `records` database:

```
use $external
db.createUser(
   {
     user: "reportingapp@EXAMPLE.NET",
     roles: [ { role: "read", db: "records" } ]
   }
)
```

Add additional principals as needed. For every user you want to authenticate using Kerberos, you must create a corresponding user in MongoDB. For more information about creating and managing users, see `http://docs.mongodb.org/manual/reference/command/nav-user-management`.

Step 4: Start `mongod.exe` with Kerberos support. You must start `mongod.exe` as the *service principal account* (page 61).

To start `mongod.exe` with Kerberos support, set the `mongod.exe` parameter `authenticationMechanisms` to `GSSAPI`:

```
mongod.exe --setParameter authenticationMechanisms=GSSAPI <additional mongod.exe options>
```

For example, the following starts a standalone `mongod.exe` instance with Kerberos support:

```
mongod.exe --auth --setParameter authenticationMechanisms=GSSAPI
```

Modify or include additional `mongod.exe` options as required for your configuration.

Step 5: Connect `mongo.exe` shell to `mongod.exe` and authenticate. Connect the `mongo.exe` shell client as the Kerberos principal `application@EXAMPLE.NET`.

You can connect and authenticate from the command line.

```
mongo.exe --authenticationMechanism=GSSAPI --authenticationDatabase='$external' \
--username reportingapp@EXAMPLE.NET
```

Or, alternatively, you can first connect `mongo.exe` to the `mongod.exe`, and then from the `mongo.exe` shell, use the `db.auth()` method to authenticate in the `$external` database.

```
use $external
db.auth( { mechanism: "GSSAPI", user: "reportingapp@EXAMPLE.NET" } )
```

Additional Considerations

Configure `mongos.exe` for Kerberos To start `mongos.exe` with Kerberos support, set the `mongos.exe` parameter `authenticationMechanisms` to `GSSAPI`. You must start `mongos.exe` as the *service principal account* (page 61).:

```
mongos.exe --setParameter authenticationMechanisms=GSSAPI <additional mongos options>
```

For example, the following starts a `mongos` instance with Kerberos support:

```
mongos.exe --setParameter authenticationMechanisms=GSSAPI --configdb shard0.example.net, shard1.examp
```

Modify or include any additional `mongos.exe` options as required for your configuration. For example, instead of using *`--keyFile`* for internal authentication of sharded cluster members, you can use *x.509 member authentication* (page 46) instead.

Assign Service Principal Name to MongoDB Windows Service Use `setspn.exe` to assign the service principal name (SPN) to the account running the `mongod.exe` and the `mongos.exe` service:

```
setspn.exe -A <service>/<fully qualified domain name> <service account name>
```

For example, if `mongod.exe` runs as a service named `mongodb` on `testserver.mongodb.com` with the service account name `mongodtest`, assign the SPN as follows:

```
setspn.exe -A mongodb/testserver.mongodb.com mongodtest
```

Incorporate Additional Authentication Mechanisms Kerberos authentication (*GSSAPI* (page 8) (Kerberos)) can work alongside MongoDB's challenge/response authentication mechanisms (*SCRAM-SHA-1* (page 6) and *MONGODB-CR* (page 7)), MongoDB's authentication mechanism for LDAP (*PLAIN* (page 8) (LDAP SASL)), and MongoDB's authentication mechanism for x.509 (*MONGODB-X509* (page 8)). Specify the mechanisms as follows:

```
--setParameter authenticationMechanisms=GSSAPI,SCRAM-SHA-1
```

Only add the other mechanisms if in use. This parameter setting does not affect MongoDB's internal authentication of cluster members.

Authenticate to a MongoDB Instance or Cluster

Overview

To authenticate to a running `mongod` or `mongos` instance, you must have user credentials for a resource on that instance. When you authenticate to MongoDB, you authenticate either to a database or to a cluster. Your user privileges determine the resource you can authenticate to.

You authenticate to a resource either by:

- using the authentication options when connecting to the `mongod` or `mongos` instance, or
- connecting first and then authenticating to the resource with the `authenticate` command or the `db.auth()` method.

This section describes both approaches.

In general, always use a trusted channel (VPN, TLS/SSL, trusted wired network) for connecting to a MongoDB instance.

Prerequisites

You must have user credentials on the database or cluster to which you are authenticating.

Procedures

Authenticate When First Connecting to MongoDB

Step 1: Specify your credentials when starting the `mongo` instance. When using `mongo` to connect to a `mongod` or `mongos`, enter your `username`, `password`, and *`authenticationDatabase`*. For example:

```
mongo --username "prodManager" --password "cleartextPassword" --authenticationDatabase "products"
```

Step 2: Close the session when your work is complete. To close an authenticated session, use the `logout` command.:

```
db.runCommand( { logout: 1 } )
```

Authenticate After Connecting to MongoDB

Step 1: Connect to a MongoDB instance. Connect to a `mongod` or `mongos` instance.

Step 2: Switch to the database to which to authenticate.

```
use <database>
```

Step 3: Authenticate. Use either the `authenticate` command or the `db.auth()` method to provide your username and password to the database. For example:

```
db.auth( "prodManager", "cleartextPassword" )
```

Step 4: Close the session when your work is complete. To close an authenticated session, use the `logout` command.:

```
db.runCommand( { logout: 1 } )
```

Generate a Key File

Overview

This section describes how to generate a key file to store authentication information. After generating a key file, specify the key file using the `keyFile` option when starting a `mongod` or `mongos` instance.

A key's length must be between 6 and 1024 characters and may only contain characters in the base64 set. The key file must not have group or world permissions on UNIX systems. Key file permissions are not checked on Windows systems.

MongoDB strips whitespace characters (e.g. `x0d`, `x09`, and `x20`) for cross-platform convenience. As a result, the following operations produce identical keys:

```
echo -e "my secret key" > key1
echo -e "my secret key\n" > key2
echo -e "my    secret    key" > key3
echo -e "my\r\nsecret\r\nkey\r\n" > key4
```

Procedure

Step 1: Create a key file. Create the key file your deployment will use to authenticate servers to each other.

To generate pseudo-random data to use for a `keyfile`, issue the following `openssl` command:

```
openssl rand -base64 741 > mongodb-keyfile
chmod 600 mongodb-keyfile
```

You may generate a key file using any method you choose. Always ensure that the password stored in the key file is both long and contains a high amount of entropy. Using `openssl` in this manner helps generate such a key.

Step 2: Specify the key file when starting a MongoDB instance. Specify the path to the key file with the `keyFile` option.

Troubleshoot Kerberos Authentication on Linux

New in version 2.4.

Kerberos Configuration Checklist

If you have difficulty starting `mongod` or `mongos` with *Kerberos* (page 11) on Linux systems, ensure that:

- The `mongod` and the `mongos` binaries are from MongoDB Enterprise.

  To verify MongoDB Enterprise binaries:

  ```
  mongod --version
  ```

  In the output from this command, look for the string `modules: subscription` or `modules: enterprise` to confirm your system has MongoDB Enterprise.

- You are not using the HTTP Console[73]. MongoDB Enterprise does not support Kerberos authentication over the HTTP Console interface.

- Either the service principal name (SPN) in the *keytab file* (page 12) matches the SPN for the `mongod` or `mongos` instance, or the `mongod` or the `mongos` instance use the `--setParameter saslHostName=<host name>` to match the name in the keytab file.

- The canonical system hostname of the system that runs the `mongod` or `mongos` instance is a resolvable, fully qualified domain for this host. You can test the system hostname resolution with the `hostname -f` command at the system prompt.

- Each host that runs a `mongod` or `mongos` instance has both the `A` and `PTR` DNS records to provide forward and reverse lookup. The records allow the host to resolve the components of the Kerberos infrastructure.

- Both the Kerberos Key Distribution Center (KDC) and the system running `mongod` instance or `mongos` must be able to resolve each other using DNS. By default, Kerberos attempts to resolve hosts using the content of the `/etc/kerb5.conf` before using DNS to resolve hosts.

- The time synchronization of the systems running `mongod` or the `mongos` instances and the Kerberos infrastructure are within the maximum time skew (default is 5 minutes) of each other. Time differences greater than the maximum time skew will prevent successful authentication.

Debug with More Verbose Logs

If you still encounter problems with Kerberos on Linux, you can start both `mongod` and `mongo` (or another client) with the environment variable `KRB5_TRACE` set to different files to produce more verbose logging of the Kerberos process to help further troubleshooting. For example, the following starts a standalone `mongod` with `KRB5_TRACE` set:

```
env KRB5_KTNAME=/opt/mongodb/mongod.keytab \
    KRB5_TRACE=/opt/mongodb/log/mongodb-kerberos.log \
    /opt/mongodb/bin/mongod --dbpath /opt/mongodb/data \
    --fork --logpath /opt/mongodb/log/mongod.log \
    --auth --setParameter authenticationMechanisms=GSSAPI
```

[73] https://docs.mongodb.org/ecosystem/tools/http-interface/#http-console

Common Error Messages

In some situations, MongoDB will return error messages from the GSSAPI interface if there is a problem with the Kerberos service. Some common error messages are:

`GSSAPI error in client while negotiating security context.` This error occurs on the client and reflects insufficient credentials or a malicious attempt to authenticate.

If you receive this error, ensure that you are using the correct credentials and the correct fully qualified domain name when connecting to the host.

`GSSAPI error acquiring credentials.` This error occurs during the start of the `mongod` or `mongos` and reflects improper configuration of the system hostname or a missing or incorrectly configured keytab file.

If you encounter this problem, consider the items in the *Kerberos Configuration Checklist* (page 63), in particular, whether the SPN in the *keytab file* (page 12) matches the SPN for the `mongod` or `mongos` instance.

To determine whether the SPNs match:

1. Examine the keytab file, with the following command:

   ```
   klist -k <keytab>
   ```

   Replace `<keytab>` with the path to your keytab file.

2. Check the configured hostname for your system, with the following command:

   ```
   hostname -f
   ```

   Ensure that this name matches the name in the keytab file, or start `mongod` or `mongos` with the `--setParameter saslHostName=<hostname>`.

See also:

- *Kerberos Authentication* (page 11)
- *Configure MongoDB with Kerberos Authentication on Linux* (page 55)
- *Configure MongoDB with Kerberos Authentication on Windows* (page 59)

Implement Field Level Redaction

The `$redact` pipeline operator restricts the contents of the documents based on information stored in the documents themselves.

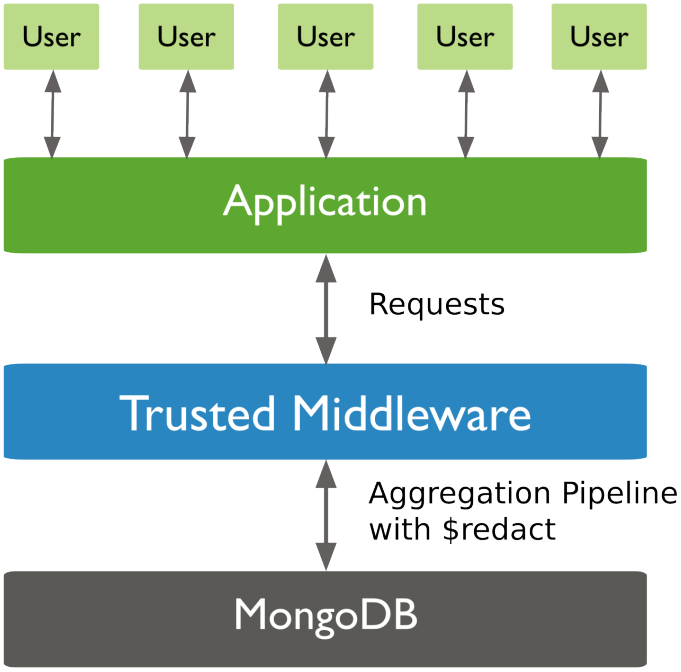
To store the access criteria data, add a field to the documents and embedded documents. To allow for multiple combinations of access levels for the same data, consider setting the access field to an array of arrays. Each array element contains a required set that allows a user with that set to access the data.

Then, include the `$redact` stage in the `db.collection.aggregate()` operation to restrict contents of the result set based on the access required to view the data.

For more information on the `$redact` pipeline operator, including its syntax and associated system variables as well as additional examples, see `$redact`.

Procedure

For example, a `forecasts` collection contains documents of the following form where the `tags` field determines the access levels required to view the data:




```
{
   _id: 1,
   title: "123 Department Report",
   tags: [ [ "G" ], [ "FDW" ] ],
   year: 2014,
   subsections: [
       {
           subtitle: "Section 1: Overview",
           tags: [ [ "SI", "G" ], [ "FDW" ] ],
           content:  "Section 1: This is the content of section 1."
       },
       {
           subtitle: "Section 2: Analysis",
           tags: [ [ "STLW" ] ],
           content: "Section 2: This is the content of section 2."
       },
       {
           subtitle: "Section 3: Budgeting",
           tags: [ [ "TK" ], [ "FDW", "TGE" ] ],
           content: {
               text: "Section 3: This is the content of section3.",
               tags: [ [ "HCS"], [ "FDW", "TGE", "BX" ] ]
           }
       }
   ]
}
```

For each document, the `tags` field contains various access groupings necessary to view the data. For example, the value `[ [ "G" ], [ "FDW", "TGE" ] ]` can specify that a user requires either access level `["G"]` or both `[`

"FDW", "TGE"] to view the data.

Consider a user who only has access to view information tagged with either "FDW" or "TGE". To run a query on all documents with year 2014 for this user, include a $redact stage as in the following:

```
var userAccess = [ "FDW", "TGE" ];
db.forecasts.aggregate(
   [
     { $match: { year: 2014 } },
     { $redact:
         {
           $cond: {
                    if: { $anyElementTrue:
                           {
                             $map: {
                                     input: "$tags" ,
                                     as: "fieldTag",
                                     in: { $setIsSubset: [ "$$fieldTag", userAccess ] }
                                   }
                           }
                       },
                     then: "$$DESCEND",
                     else: "$$PRUNE"
                  }
         }
     }
   ]
)
```

The aggregation operation returns the following "redacted" document for the user:

```
{ "_id" : 1,
  "title" : "123 Department Report",
  "tags" : [ [ "G" ], [ "FDW" ] ],
  "year" : 2014,
  "subsections" :
     [
        {
          "subtitle" : "Section 1: Overview",
          "tags" : [ [ "SI", "G" ], [ "FDW" ] ],
          "content" : "Section 1: This is the content of section 1."
        },
       {
         "subtitle" : "Section 3: Budgeting",
         "tags" : [ [ "TK" ], [ "FDW", "TGE" ] ]
       }
     ]
}
```

See also:

$map, $setIsSubset, $anyElementTrue

3.4 User and Role Management Tutorials

The following tutorials provide instructions on how to enable authentication and limit access for users with privilege roles.

Create a User Administrator **(page 67)** Create users with special permissions to create, modify, and remove other users, as well as administer authentication credentials (e.g. passwords).

Manage User and Roles **(page 69)** Manage users by creating new users, creating new roles, and modifying existing users.

Change Your Password and Custom Data **(page 75)** Users with sufficient access can change their own passwords and modify the optional *custom data* associated with their user credential.

Create an Administrative User with Unrestricted Access **(page 77)** Create a user with unrestricted access. Create such a user only in unique situations. In general, all users in the system should have no more access than needed to perform their required operations.

Create a User Administrator

Overview

User administrators create users and create and assigns roles. A user administrator can grant any privilege in the database and can create new ones. In a MongoDB deployment, create the user administrator as the first user. Then let this user create all other users.

To provide user administrators, MongoDB has `userAdmin` (page 87) and `userAdminAnyDatabase` (page 92) roles, which grant access to *actions* (page 99) that support user and role management. Following the policy of *least privilege* `userAdmin` (page 87) and `userAdminAnyDatabase` (page 92) confer no additional privileges.

Carefully control access to these roles. A user with either of these roles can grant *itself* unlimited additional privileges. Specifically, a user with the `userAdmin` (page 87) role can grant itself any privilege in the database. A user assigned either the `userAdmin` (page 87) role on the `admin` database or the `userAdminAnyDatabase` (page 92) can grant itself any privilege *in the system*.

Prerequisites

Required Access

- To create a new user in a database, you must have `createUser` (page 100) *action* (page 99) on that *database resource* (page 98).
- To grant roles to a user, you must have the `grantRole` (page 100) *action* (page 99) on the role's database.

Built-in roles `userAdmin` (page 87) and `userAdminAnyDatabase` (page 92) provide `createUser` (page 100) and `grantRole` (page 100) actions on their respective *resources* (page 97).

First User Restrictions If your MongoDB deployment has no users, you *must* connect to `mongod` using the *localhost exception* (page 9) or use the `--noauth` option when starting `mongod` to gain full access the system. Once you have access, you can skip to *Creating the system user administrator* in this procedure.

If users exist in the MongoDB database, but none of them has the appropriate prerequisites to create a new user or you do not have access to them, you *must* restart `mongod` with the `--noauth` option.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` either through the *localhost exception* (page 9) or as a user with the privileges indicated in the prerequisites section.

In the following example, `manager` has the required privileges specified in *Prerequisites* (page 67).

```
mongo --port 27017 -u manager -p 123456 --authenticationDatabase admin
```

Step 2: Create the system user administrator. Add the user with the `userAdminAnyDatabase` (page 92) role, and only that role.

The following example creates the user `siteUserAdmin` user on the `admin` database:

```
use admin
db.createUser(
  {
    user: "siteUserAdmin",
    pwd: "password",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

Step 3: Create a user administrator for a single database. Optionally, you may want to create user administrators that only have access to administer users in a specific database by way of the `userAdmin` (page 87) role.

The following example creates the user `recordsUserAdmin` on the `records` database:

```
use records
db.createUser(
  {
    user: "recordsUserAdmin",
    pwd: "password",
    roles: [ { role: "userAdmin", db: "records" } ]
  }
)
```

Related Documents

- *Authentication* (page 6)
- *Security Introduction* (page 3)
- *Enable Client Access Control* (page 40)
- *Authentication Tutorials* (page 39)

Additional Resources

- Security Architecture White Paper[74]
- Webinar: Securing Your MongoDB Deployment[75]
- Creating a Single View Part 3: Securing Your Deployment[76]

[74] https://www.mongodb.com/lp/white-paper/mongodb-security-architecture?jmp=docs
[75] http://www.mongodb.com/webinar/securing-your-mongodb-deployment?jmp=docs
[76] https://www.mongodb.com/presentations/creating-single-view-part-3-securing-your-deployment?jmp=docs

Manage User and Roles

Overview

Changed in version 2.6: MongoDB 2.6 introduces a new *authorization model* (page 14).

MongoDB employs Role-Based Access Control (RBAC) to determine access for users. A user is granted one or more *roles* (page 14) that determine the user's access or privileges to MongoDB *resources* (page 97) and the *actions* (page 99) that user can perform. A user should have only the minimal set of privileges required to ensure a system of *least privilege*.

Each application and user of a MongoDB system should map to a distinct application or administrator. This *access isolation* facilitates access revocation and ongoing user maintenance.

This tutorial provides examples for user and role management under the MongoDB's authorization model.

Prerequisites

Important: If you have enabled *authorization* (page 14) for your deployment, you must authenticate as a user with the required privileges specified in each section. A *user administrator* (page 67) with the `userAdminAnyDatabase` (page 92) role, or `userAdmin` (page 87) role in the specific databases, provides the required privileges to perform the operations listed in this tutorial.

If you have not yet created a user administrator, do so as described in *Create a User Administrator* (page 67).

Add a User

To create a user, specify the user name, password, and *roles* (page 14). For users that authenticate using external mechanisms [77], you do not need to provide the password when creating users.

When assigning roles, select the roles that have the exact required *privileges* (page 15). If the correct roles does not exist, you can *create new roles* (page 70).

Prerequisites

- To create a new user in a database, you must have `createUser` (page 100) *action* (page 99) on that *database resource* (page 98).
- To grant roles to a user, you must have the `grantRole` (page 100) *action* (page 99) on the role's database.

Built-in roles `userAdmin` (page 87) and `userAdminAnyDatabase` (page 92) provide `createUser` (page 100) and `grantRole` (page 100) actions on their respective *resources* (page 97).

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` as a user with the privileges specified in the prerequisite section.

The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

[77] See *x.509 Certificate Authentication* (page 8), *Kerberos Authentication* (page 8), and *LDAP Proxy Authority Authentication* (page 8)

Step 2: Create the new user. Create the user in the database to which the user will belong. Pass a well formed user document to the `db.createUser()` method.

The following operation creates a user in the `reporting` database with the specified name, password, and roles.

```
use reporting
db.createUser(
    {
      user: "reportsUser",
      pwd: "12345678",
      roles: [
         { role: "read", db: "reporting" },
         { role: "read", db: "products" },
         { role: "read", db: "sales" },
         { role: "readWrite", db: "accounts" }
      ]
    }
)
```

To authenticate the `reportsUser`, you must authenticate the user in the `reporting` database; i.e. specify *`--authenticationDatabase reporting`*.

You can create a user without assigning roles, choosing instead to assign the roles later. To do so, create the user with an empty `roles` (page 96) array.

Create a User-Defined Role

Roles grant users access to MongoDB resources. MongoDB provides a number of *built-in roles* (page 85) that administrators can use to control access to a MongoDB system. However, if these roles cannot describe the desired set of privileges, you can create new roles in a particular database.

Except for roles created in the `admin` database, a role can only include privileges that apply to its database and can only inherit from other roles in its database.

A role created in the `admin` database can include privileges that apply to the `admin` database, other databases or to the *cluster* (page 98) resource, and can inherit from roles in other databases as well as the `admin` database.

To create a new role, use the `db.createRole()` method, specifying the privileges in the `privileges` array and the inherited roles in the `roles` array.

MongoDB uses the combination of the database name and the role name to uniquely define a role. Each role is scoped to the database in which you create the role, but MongoDB stores all role information in the `admin.system.roles` collection in the `admin` database.

Prerequisites To create a role in a database, you must have:

- the `createRole` (page 100) *action* (page 99) on that *database resource* (page 98).
- the `grantRole` (page 100) *action* (page 99) on that database to specify privileges for the new role as well as to specify roles to inherit from.

Built-in roles `userAdmin` (page 87) and `userAdminAnyDatabase` (page 92) provide `createRole` (page 100) and `grantRole` (page 100) actions on their respective *resources* (page 97).

Create a Role to Manage Current Operations The following example creates a role named `manageOpRole` which provides only the privileges to run both `db.currentOp()` and `db.killOp()`.[78]

[78] The built-in role `clusterMonitor` (page 89) also provides the privilege to run `db.currentOp()` along with other privileges, and the built-in role `hostManager` (page 90) provides the privilege to run `db.killOp()` along with other privileges.

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` with the privileges specified in the *Prerequisites* (page 70) section.

The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

The `siteUserAdmin` has privileges to create roles in the `admin` as well as other databases.

Step 2: Create a new role to manage current operations. `manageOpRole` has privileges that act on multiple databases as well as the *cluster resource* (page 98). As such, you must create the role in the `admin` database.

```
use admin
db.createRole(
   {
     role: "manageOpRole",
     privileges: [
       { resource: { cluster: true }, actions: [ "killop", "inprog" ] },
       { resource: { db: "", collection: "" }, actions: [ "killCursors" ] }
     ],
     roles: []
   }
)
```

The new role grants permissions to kill any operations.

> **Warning:** Terminate running operations with extreme caution. Only use `db.killOp()` to terminate operations initiated by clients and *do not* terminate internal database operations.

Create a Role to Run `mongostat` The following example creates a role named `mongostatRole` that provides only the privileges to run `mongostat`. [79]

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` with the privileges specified in the *Prerequisites* (page 70) section.

The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

The `siteUserAdmin` has privileges to create roles in the `admin` as well as other databases.

Step 2: Create a new role to manage current operations. `mongostatRole` has privileges that act on the *cluster resource* (page 98). As such, you must create the role in the `admin` database.

```
use admin
db.createRole(
   {
     role: "mongostatRole",
     privileges: [
       { resource: { cluster: true }, actions: [ "serverStatus" ] }
     ],
     roles: []
   }
)
```

[79] The built-in role `clusterMonitor` (page 89) also provides the privilege to run `mongostat` along with other privileges.

Modify Access for Existing User

Prerequisites

- You must have the `grantRole` (page 100) *action* (page 99) on a database to grant a role on that database.
- You must have the `revokeRole` (page 100) *action* (page 99) on a database to revoke a role on that database.
- To view a role's information, you must be either explicitly granted the role or must have the `viewRole` (page 100) *action* (page 99) on the role's database.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` as a user with the privileges specified in the prerequisite section.

The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

Step 2: Identify the user's roles and privileges. To display the roles and privileges of the user to be modified, use the `db.getUser()` and `db.getRole()` methods.

For example, to view roles for `reportsUser` created in *Add a User* (page 69), issue:

```
use reporting
db.getUser("reportsUser")
```

To display the privileges granted to the user by the `readWrite` role on the `"accounts"` database, issue:

```
use accounts
db.getRole( "readWrite", { showPrivileges: true } )
```

Step 3: Identify the privileges to grant or revoke. If the user requires additional privileges, grant to the user the role, or roles, with the required set of privileges. If such a role does not exist, *create a new role* (page 70) with the appropriate set of privileges.

To revoke a subset of privileges provided by an existing role: revoke the original role and grant a role that contains only the required privileges. You may need to *create a new role* (page 70) if a role does not exist.

Step 4: Modify the user's access.

Revoke a Role Revoke a role with the `db.revokeRolesFromUser()` method. The following example operation removes the `readWrite` (page 86) role on the `accounts` database from the `reportsUser`:

```
use reporting
db.revokeRolesFromUser(
    "reportsUser",
    [
      { role: "readWrite", db: "accounts" }
    ]
)
```

Grant a Role Grant a role using the `db.grantRolesToUser()` method. For example, the following operation grants the `reportsUser` user the `read` (page 85) role on the `accounts` database:

```
use reporting
db.grantRolesToUser(
    "reportsUser",
    [
      { role: "read", db: "accounts" }
    ]
)
```

For sharded clusters, the changes to the user are instant on the `mongos` on which the command runs. However, for other `mongos` instances in the cluster, the user cache may wait up to 10 minutes to refresh. See `userCacheInvalidationIntervalSecs`.

Modify Password for Existing User

Prerequisites To modify the password of another user on a database, you must have the `changeAnyPassword` *action* (page 99) on that database.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to the `mongod` or `mongos` with the privileges specified in the *Prerequisites* (page 73) section.

The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

Step 2: Change the password. Pass the user's username and the new password to the `db.changeUserPassword()` method.

The following operation changes the `reporting` user's password to `SOh3TbYhxuLiW8ypJPxmt1oOfL`:

```
db.changeUserPassword("reporting", "SOh3TbYhxuLiW8ypJPxmt1oOfL")
```

See also:

Change Your Password and Custom Data (page 75)

View a User's Role

Prerequisites To view another user's information, you must have the `viewUser` (page 100) *action* (page 99) on the other user's database.

Users can view their own information.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` as a user with the privileges specified in the prerequisite section.

The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

Step 2: Identify the user's roles. Use the `usersInfo` command or `db.getUser()` method to display user information.

For example, to view roles for `reportsUser` created in *Add a User* (page 69), issue:

```
use reporting
db.getUser("reportsUser")
```

In the returned document, the `roles` (page 96) field displays all roles for `reportsUser`:

```
...
"roles" : [
   { "role" : "readWrite", "db" : "accounts" },
   { "role" : "read", "db" : "reporting" },
   { "role" : "read", "db" : "products" },
   { "role" : "read", "db" : "sales" }
]
```

View Role's Privileges

Prerequisites To view a role's information, you must be either explicitly granted the role or must have the `viewRole` (page 100) *action* (page 99) on the role's database.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to `mongod` or `mongos` as a user with the privileges specified in the prerequisite section.

The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

Step 2: Identify the privileges granted by a role. For a given role, use the `db.getRole()` method, or the `rolesInfo` command, with the `showPrivileges` option:

For example, to view the privileges granted by `read` role on the `products` database, use the following operation, issue:

```
use products
db.getRole( "read", { showPrivileges: true } )
```

In the returned document, the `privileges` and `inheritedPrivileges` arrays. The `privileges` lists the privileges directly specified by the role and excludes those privileges inherited from other roles. The `inheritedPrivileges` lists all privileges granted by this role, both directly specified and inherited. If the role does not inherit from other roles, the two fields are the same.

```
...
"privileges" : [
  {
    "resource": { "db" : "products", "collection" : "" },
    "actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]
  },
  {
    "resource" : { "db" : "products", "collection" : "system.js" },
    "actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]
  }
],
"inheritedPrivileges" : [
  {
    "resource": { "db" : "products", "collection" : "" },
    "actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]
  },
  {
    "resource" : { "db" : "products", "collection" : "system.js" },
    "actions": [ "collStats","dbHash","dbStats","find","killCursors","planCacheRead" ]
  }
]
```

Change Your Password and Custom Data

Changed in version 2.6.

Overview

Users with appropriate privileges can change their own passwords and custom data. `Custom data` (page 97) stores optional user information.

Considerations

To generate a strong password for use in this procedure, you can use the `openssl` utility's `rand` command. For example, issue `openssl rand` with the following options to create a base64-encoded string of 48 pseudo-random bytes:

```
openssl rand -base64 48
```

Prerequisites

To modify your own password and custom data, you must have privileges that grant `changeOwnPassword` (page 99) and `changeOwnCustomData` (page 99) *actions* (page 99) respectively on the user's database.

Step 1: Connect as a user with privileges to manage users and roles. Connect to the `mongod` or `mongos` with privileges to manage users and roles, such as a user with `userAdminAnyDatabase` (page 92) role. The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

Step 2: Create a role with appropriate privileges. In the `admin` database, `create` a new role with `changeOwnPassword` (page 99) and `changeOwnCustomData` (page 99).

```
use admin
db.createRole(
   { role: "changeOwnPasswordCustomDataRole",
     privileges: [
        {
          resource: { db: "", collection: ""},
          actions: [ "changeOwnPassword", "changeOwnCustomData" ]
        }
     ],
     roles: []
   }
)
```

Step 3: Add a user with this role. In the `test` database, `create` a new user with the created `"changeOwnPasswordCustomDataRole"` role. For example, the following operation creates a user with both the built-in role `readWrite` (page 86) and the user-created `"changeOwnPasswordCustomDataRole"`.

```
use test
db.createUser(
   {
     user:"user123",
     pwd:"12345678",
     roles:[ "readWrite", { role:"changeOwnPasswordCustomDataRole", db:"admin" } ]
   }
)
```

To grant an existing user the new role, use `db.grantRolesToUser()`.

Procedure

Step 1: Connect with the appropriate privileges. Connect to the `mongod` or `mongos` as a user with appropriate privileges.

For example, the following operation connects to MongoDB as `user123` created in the *Prerequisites* (page 75) section.

```
mongo --port 27017 -u user123 -p 12345678 --authenticationDatabase test
```

To check that you have the privileges specified in the *Prerequisites* (page 75) section as well as to see user information, use the `usersInfo` command with the `showPrivileges` option.

Step 2: Change your password and custom data. Use the `db.updateUser()` method to update the password and custom data.

For example, the following operation changes thw user's password to `KNlZmiaNUp0B` and custom data to `{ title: "Senior Manager" }`:

```
use test
db.updateUser(
   "user123",
   {
      pwd: "KNlZmiaNUp0B",
      customData: { title: "Senior Manager" }
```

```
    }
)
```

Create an Administrative User with Unrestricted Access

Overview

Most users should have only the minimal set of privileges required for their operations, in keeping with the policy of *least privilege*. However, some authorization architectures may require a user with unrestricted access. To support these *super users*, you can create users with access to all database *resources* (page 97) and *actions* (page 99).

For many deployments, you may be able to avoid having *any* users with unrestricted access by having an administrative user with the `createUser` (page 100) and `grantRole` (page 100) actions granted as needed to support operations.

If users truly need unrestricted access to a MongoDB deployment, MongoDB provides a *built-in role* (page 85) named `root` (page 93) that grants the combined privileges of all built-in roles. This document describes how to create an administrative user with the `root` (page 93) role.

For descriptions of the access each built-in role provides, see the section on *built-in roles* (page 85).

Prerequisites

Required Access

- To create a new user in a database, you must have `createUser` (page 100) *action* (page 99) on that *database resource* (page 98).
- To grant roles to a user, you must have the `grantRole` (page 100) *action* (page 99) on the role's database.

Built-in roles `userAdmin` (page 87) and `userAdminAnyDatabase` (page 92) provide `createUser` (page 100) and `grantRole` (page 100) actions on their respective *resources* (page 97).

First User Restrictions If your MongoDB deployment has no users, you *must* connect to `mongod` using the *localhost exception* (page 9) or use the `--noauth` option when starting `mongod` to gain full access the system. Once you have access, you can skip to *Creating the system user administrator* in this procedure.

If users exist in the MongoDB database, but none of them has the appropriate prerequisites to create a new user or you do not have access to them, you *must* restart `mongod` with the `--noauth` option.

Procedure

Step 1: Connect to MongoDB with the appropriate privileges. Connect to the `mongod` or `mongos` as a user with the privileges specified in the *Prerequisites* (page 77) section.

The following procedure uses the `siteUserAdmin` created in *Create a User Administrator* (page 67).

```
mongo --port 27017 -u siteUserAdmin -p password --authenticationDatabase admin
```

Step 2: Create the administrative user. In the `admin` database, create a new user using the `db.createUser()` method. Give the user the built-in `root` (page 93) role.

For example:

```
use admin
db.createUser(
    {
      user: "superuser",
      pwd: "12345678",
      roles: [ "root" ]
    }
)
```

Authenticate against the `admin` database to test the new user account. Use `db.auth()` while using the `admin` database or use the `mongo` shell with the `--authenticationDatabase` option.

3.5 Auditing Tutorials

The following tutorials provide instructions on how to enable auditing for system events and specify which events to audit.

Configure System Events Auditing **(page 78)** Enable and configure MongoDB Enterprise system event auditing feature.

Configure Audit Filters **(page 80)** Specify which events to audit.

Configure System Events Auditing

New in version 2.6.

MongoDB Enterprise[80] supports *auditing* (page 16) of various operations. A complete auditing solution must involve **all** `mongod` server and `mongos` router processes.

The audit facility can write audit events to the console, the *syslog* (option is unavailable on Windows), a JSON file, or a BSON file. For details on the audited operations and the audit log messages, see *System Event Audit Messages* (page 104).

Enable and Configure Audit Output

Use the `--auditDestination` option to enable auditing and specify where to output the audit events.

> **Warning:** For sharded clusters, if you enable auditing on `mongos` instances, you must enable auditing on all `mongod` instances in the cluster, i.e. shards and config servers.

Output to Syslog To enable auditing and print audit events to the syslog (option is unavailable on Windows) in JSON format, specify `syslog` for the `--auditDestination` setting. For example:

```
mongod --dbpath data/db --auditDestination syslog
```

> **Warning:** The syslog message limit can result in the truncation of the audit messages. The auditing system will neither detect the truncation nor error upon its occurrence.

You may also specify these options in the `configuration file`:

[80] https://www.mongodb.com/products/mongodb-enterprise-advanced?jmp=docs

```
storage:
   dbPath: data/db
auditLog:
   destination: syslog
```

Output to Console To enable auditing and print the audit events to standard output (i.e. `stdout`), specify `console` for the *`--auditDestination`* setting. For example:

```
mongod --dbpath data/db --auditDestination console
```

You may also specify these options in the `configuration file`:

```
storage:
   dbPath: data/db
auditLog:
   destination: console
```

Output to JSON File To enable auditing and print audit events to a file in JSON format, specify `file` for the *`--auditDestination`* setting, `JSON` for the *`--auditFormat`* setting, and the output filename for the *`--auditPath`*. The *`--auditPath`* option accepts either full path name or relative path name. For example, the following enables auditing and records audit events to a file with the relative path name of `data/db/auditLog.json`:

```
mongod --dbpath data/db --auditDestination file --auditFormat JSON --auditPath data/db/auditLog.json
```

The audit file rotates at the same time as the server log file.

You may also specify these options in the `configuration file`:

```
storage:
   dbPath: data/db
auditLog:
   destination: file
   format: JSON
   path: data/db/auditLog.json
```

Note: Printing audit events to a file in JSON format degrades server performance more than printing to a file in BSON format.

Output to BSON File To enable auditing and print audit events to a file in BSON binary format, specify `file` for the *`--auditDestination`* setting, `BSON` for the *`--auditFormat`* setting, and the output filename for the *`--auditPath`*. The *`--auditPath`* option accepts either full path name or relative path name. For example, the following enables auditing and records audit events to a BSON file with the relative path name of `data/db/auditLog.bson`:

```
mongod --dbpath data/db --auditDestination file --auditFormat BSON --auditPath data/db/auditLog.bson
```

The audit file rotates at the same time as the server log file.

You may also specify these options in the `configuration file`:

```
storage:
   dbPath: data/db
auditLog:
   destination: file
```

```
   format: BSON
   path: data/db/auditLog.bson
```

To view the contents of the file, pass the file to the MongoDB utility `bsondump`. For example, the following converts the audit log into a human-readable form and output to the terminal:

```
bsondump data/db/auditLog.bson
```

See also:

Configure Audit Filters (page 80), *Auditing* (page 16), *System Event Audit Messages* (page 104)

Configure Audit Filters

MongoDB Enterprise[81] supports *auditing* (page 16) of various operations. When *enabled* (page 78), the audit facility, by default, records all auditable operations as detailed in *Audit Event Actions, Details, and Results* (page 105). To specify which events to record, the audit feature includes the *--auditFilter* option.

`--auditFilter` Option

The *--auditFilter* option takes a string representation of a query document of the form:

```
{ <field1>: <expression1>, ... }
```

- The `<field>` can be *any field in the audit message* (page 104), including fields returned in the *param* (page 105) document.
- The `<expression>` is a *query condition expression*.

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

To specify the audit filter in a `configuration file`, you must use the YAML format of the configuration file.

Examples

Filter for Multiple Operation Types The following example audits only the `createCollection` (page 99) and `dropCollection` (page 100) actions by using the filter:

```
{ atype: { $in: [ "createCollection", "dropCollection" ] } }
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auditDestination file --auditFilter '{ atype: { $in: [ "createCollection",
```

To specify the audit filter in a `configuration file`, you must use the YAML format of the configuration file.

```
storage:
   dbPath: data/db
auditLog:
   destination: file
   format: BSON
   path: data/db/auditLog.bson
   filter: '{ atype: { $in: [ "createCollection", "dropCollection" ] } }'
```

[81] https://www.mongodb.com/products/mongodb-enterprise-advanced?jmp=docs

Filter on Authentication Operations on a Single Database The `<field>` can include *any field in the audit message* (page 104). For authentication operations (i.e. `atype: "authenticate"`), the audit messages include a `db` field in the `param` document.

The following example audits only the `authenticate` operations that occur against the `test` database by using the filter:

```
{ atype: "authenticate", "param.db": "test" }
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --auditDestination file --auditFilter '{ atype: "authenticate", "param
```

To specify the audit filter in a `configuration file`, you must use the YAML format of the configuration file.

```
storage:
   dbPath: data/db
security:
   authorization: enabled
auditLog:
   destination: file
   format: BSON
   path: data/db/auditLog.bson
   filter: '{ atype: "authenticate", "param.db": "test" }'
```

To filter on all `authenticate` operations across databases, use the filter `{ atype: "authenticate" }`.

Filter on Collection Creation and Drop Operations for a Single Database The `<field>` can include *any field in the audit message* (page 104). For collection creation and drop operations (i.e. `atype: "createCollection"` and `atype: "dropCollection"`), the audit messages include a namespace `ns` field in the `param` document.

The following example audits only the `createCollection` and `dropCollection` operations that occur against the `test` database by using the filter:

Note: The regular expression requires two backslashes (`\\`) to escape the dot (`.`).

```
{ atype: { $in: [ "createCollection", "dropCollection" ] }, "param.ns": /^test\\./ } }
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --auditDestination file --auditFilter '{ atype: { $in: [ "createColle
```

To specify the audit filter in a `configuration file`, you must use the YAML format of the configuration file.

```
storage:
   dbPath: data/db
security:
   authorization: enabled
auditLog:
   destination: file
   format: BSON
   path: data/db/auditLog.bson
   filter: '{ atype: { $in: [ "createCollection", "dropCollection" ] }, "param.ns": /^test\\./ } }'
```

Filter by Authorization Role The following example audits operations by users with `readWrite` (page 86) role on the `test` database, including users with roles that inherit from `readWrite` (page 86), by using the filter:

```
{ roles: { role: "readWrite", db: "test" } }
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --auditDestination file --auditFilter '{ roles: { role: "readWrite",
```

To specify the audit filter in a `configuration file`, you must use the YAML format of the configuration file.

```
storage:
   dbPath: data/db
security:
   authorization: enabled
auditLog:
   destination: file
   format: BSON
   path: data/db/auditLog.bson
   filter: '{ roles: { role: "readWrite", db: "test" } }'
```

Filter on Read and Write Operations To capture `read` and `write` operations in the audit, you must also enable the audit system to log authorization successes using the `auditAuthorizationSuccess` parameter. [82]

Note: Enabling `auditAuthorizationSuccess` degrades performance more than logging only the authorization failures.

The following example audits the `find()`, `insert()`, `remove()`, `update()`, `save()`, and `findAndModify()` operations by using the filter:

```
{ atype: "authCheck", "param.command": { $in: [ "find", "insert", "delete", "update", "findandmodify"
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --setParameter auditAuthorizationSuccess=true --auditDestination file
```

To specify the audit filter in a `configuration file`, you must use the YAML format of the configuration file.

```
storage:
   dbPath: data/db
security:
   authorization: enabled
auditLog:
   destination: file
   format: BSON
   path: data/db/auditLog.bson
   filter: '{ atype: "authCheck", "param.command": { $in: [ "find", "insert", "delete", "update", "fi
setParameter: { auditAuthorizationSuccess: true }
```

Filter on Read and Write Operations for a Collection To capture `read` and `write` operations in the audit, you must also enable the audit system to log authorization successes using the `auditAuthorizationSuccess` parameter. [1]

Note: Enabling `auditAuthorizationSuccess` degrades performance more than logging only the authorization failures.

---

[82] You can enable `auditAuthorizationSuccess` parameter without enabling `--auth`; however, all operations will return success for authorization checks.

The following example audits the `find()`, `insert()`, `remove()`, `update()`, `save()`, and `findAndModify()` operations for the collection `orders` in the database `test` by using the filter:

```
{ atype: "authCheck", "param.ns": "test.orders", "param.command": { $in: [ "find", "insert", "delete"
```

To specify an audit filter, enclose the filter document in single quotes to pass the document as a string.

```
mongod --dbpath data/db --auth --setParameter auditAuthorizationSuccess=true --auditDestination file
```

To specify the audit filter in a `configuration file`, you must use the YAML format of the configuration file.

```
storage:
   dbPath: data/db
security:
   authorization: enabled
auditLog:
   destination: file
   format: BSON
   path: data/db/auditLog.bson
   filter: '{ atype: "authCheck", "param.ns": "test.orders", "param.command": { $in: [ "find", "inse
setParameter: { auditAuthorizationSuccess: true }
```

See also:

Configure System Events Auditing (page 78), *Auditing* (page 16), *System Event Audit Messages* (page 104)

3.6 Create a Vulnerability Report

If you believe you have discovered a vulnerability in MongoDB or have experienced a security incident related to MongoDB, please report the issue to aid in its resolution.

To report an issue, we strongly suggest filing a ticket in the SECURITY[83] project in JIRA. MongoDB, Inc responds to vulnerability notifications within 48 hours.

Create the Report in JIRA

Submit a Ticket[84] in the Security[85] project on our JIRA. The ticket number will become the reference identification for the issue for its lifetime. You can use this identifier for tracking purposes.

Information to Provide

All vulnerability reports should contain as much information as possible so MongoDB's developers can move quickly to resolve the issue. In particular, please include the following:

- The name of the product.
- *Common Vulnerability* information, if applicable, including:
- CVSS (Common Vulnerability Scoring System) Score.
- CVE (Common Vulnerability and Exposures) Identifier.
- Contact information, including an email address and/or phone number, if applicable.

[83] https://jira.mongodb.org/browse/SECURITY
[84] https://jira.mongodb.org/secure/CreateIssue!default.jspa?project-field=%22Security%22
[85] https://jira.mongodb.org/browse/SECURITY

Send the Report via Email

While JIRA is the preferred reporting method, you may also report vulnerabilities via email to security@mongodb.com[86].

You may encrypt email using MongoDB's public key at https://docs.mongodb.org/10gen-security-gpg-key.asc.

MongoDB, Inc. responds to vulnerability reports sent via email with a response email that contains a reference number for a JIRA ticket posted to the SECURITY[87] project.

Evaluation of a Vulnerability Report

MongoDB, Inc. validates all submitted vulnerabilities and uses Jira to track all communications regarding a vulnerability, including requests for clarification or additional information. If needed, MongoDB representatives set up a conference call to exchange information regarding the vulnerability.

Disclosure

MongoDB, Inc. requests that you do *not* publicly disclose any information regarding the vulnerability or exploit the issue until it has had the opportunity to analyze the vulnerability, to respond to the notification, and to notify key users, customers, and partners.

The amount of time required to validate a reported vulnerability depends on the complexity and severity of the issue. MongoDB, Inc. takes all required vulnerabilities very seriously and will always ensure that there is a clear and open channel of communication with the reporter.

After validating an issue, MongoDB, Inc. coordinates public disclosure of the issue with the reporter in a mutually agreed timeframe and format. If required or requested, the reporter of a vulnerability will receive credit in the published security bulletin.

4 Security Reference

4.1 Security Methods in the `mongo` Shell

User Management and Authentication Methods

Name	Description
`db.auth()`	Authenticates a user to a database.
`db.createUser()`	Creates a new user.
`db.updateUser()`	Updates user data.
`db.changeUserPassword()`	Changes an existing user's password.
`db.removeUser()`	Deprecated. Removes a user from a database.
`db.dropAllUsers()`	Deletes all users associated with a database.
`db.dropUser()`	Removes a single user.
`db.grantRolesToUser()`	Grants a role and its privileges to a user.
`db.revokeRolesFromUser()`	Removes a role from a user.
`db.getUser()`	Returns information about the specified user.
`db.getUsers()`	Returns information about all users associated with a database.

[86] security@mongodb.com
[87] https://jira.mongodb.org/browse/SECURITY

Role Management Methods

Name	Description
`db.createRole()`	Creates a role and specifies its privileges.
`db.updateRole()`	Updates a user-defined role.
`db.dropRole()`	Deletes a user-defined role.
`db.dropAllRoles()`	Deletes all user-defined roles associated with a database.
`db.grantPrivilegesToRole()`	Assigns privileges to a user-defined role.
`db.revokePrivilegesFromRole()`	Removes the specified privileges from a user-defined role.
`db.grantRolesToRole()`	Specifies roles from which a user-defined role inherits privileges.
`db.revokeRolesFromRole()`	Removes inherited roles from a role.
`db.getRole()`	Returns information for the specified role.
`db.getRoles()`	Returns information for all the user-defined roles in a database.

4.2 Security Reference Documentation

Built-In Roles **(page 85)** Reference on MongoDB provided roles and corresponding access.

system.roles Collection **(page 93)** Describes the content of the collection that stores user-defined roles.

system.users Collection **(page 96)** Describes the content of the collection that stores users' credentials and role assignments.

Resource Document **(page 97)** Describes the resource document for roles.

Privilege Actions **(page 99)** List of the actions available for privileges.

Default MongoDB Port **(page 104)** List of default ports used by MongoDB.

System Event Audit Messages **(page 104)** Reference on system event audit messages.

Built-In Roles

MongoDB grants access to data and commands through *role-based authorization* (page 14) and provides built-in roles that provide the different levels of access commonly needed in a database system. You can additionally create *user-defined roles* (page 15).

A role grants privileges to perform sets of *actions* (page 99) on defined *resources* (page 97). A given role applies to the database on which it is defined and can grant access down to a collection level of granularity.

Each of MongoDB's built-in roles defines access at the database level for all *non*-system collections in the role's database and at the collection level for all `system collections`.

MongoDB provides the built-in *database user* (page 85) and *database administration* (page 86) roles on *every* database. MongoDB provides all other built-in roles only on the `admin` database.

This section describes the privileges for each built-in role. You can also view the privileges for a built-in role at any time by issuing the `rolesInfo` command with the `showPrivileges` and `showBuiltinRoles` fields both set to `true`.

Database User Roles

Every database includes the following client roles:

`read`
: Provides the ability to read data on all *non*-system collections and on the following system collections:

system.indexes, system.js, and system.namespaces collections. The role provides read access by granting the following *actions* (page 99):

- collStats (page 103)
- dbHash (page 103)
- dbStats (page 103)
- find (page 99)
- killCursors (page 100)
- listIndexes (page 103)
- listCollections (page 103)

readWrite

Provides all the privileges of the read (page 85) role plus ability to modify data on all *non*-system collections and the system.js collection. The role provides the following actions on those collections:

- collStats (page 103)
- convertToCapped (page 102)
- createCollection (page 99)
- dbHash (page 103)
- dbStats (page 103)
- dropCollection (page 100)
- createIndex (page 99)
- dropIndex (page 102)
- emptycapped (page 100)
- find (page 99)
- insert (page 99)
- killCursors (page 100)
- listIndexes (page 103)
- listCollections (page 103)
- remove (page 99)
- renameCollectionSameDB (page 102)
- update (page 99)

Database Administration Roles

Every database includes the following database administration roles:

dbAdmin

Provides the following *actions* (page 99) on the database's system.indexes, system.namespaces, and system.profile collections:

- collStats (page 103)
- dbHash (page 103)

- dbStats (page 103)
- find (page 99)
- killCursors (page 100)
- listIndexes (page 103)
- listCollections (page 103)
- dropCollection (page 100) and createCollection (page 99) on system.profile *only*

Changed in version 2.6.4: dbAdmin (page 86) added the createCollection (page 99) for the system.profile collection. Previous versions only had the dropCollection (page 100) on the system.profile collection.

Provides the following actions on all *non*-system collections. This role *does not* include full read access on non-system collections:

- collMod (page 102)
- collStats (page 103)
- compact (page 102)
- convertToCapped (page 102)
- createCollection (page 99)
- createIndex (page 99)
- dbStats (page 103)
- dropCollection (page 100)
- dropDatabase (page 102)
- dropIndex (page 102)
- enableProfiler (page 100)
- indexStats (page 103)
- reIndex (page 102)
- renameCollectionSameDB (page 102)
- repairDatabase (page 103)
- storageDetails (page 101)
- validate (page 104)

dbOwner

The database owner can perform any administrative action on the database. This role combines the privileges granted by the readWrite (page 86), dbAdmin (page 86) and userAdmin (page 87) roles.

userAdmin

Provides the ability to create and modify roles and users on the current database. This role also indirectly provides *superuser* (page 92) access to either the database or, if scoped to the admin database, the cluster. The userAdmin (page 87) role allows users to grant any user any privilege, including themselves.

The userAdmin (page 87) role explicitly provides the following actions:

- changeCustomData (page 99)
- changePassword (page 99)
- createRole (page 100)

- createUser (page 100)
- dropRole (page 100)
- dropUser (page 100)
- grantRole (page 100)
- revokeRole (page 100)
- viewRole (page 100)
- viewUser (page 100)

Cluster Administration Roles

The admin database includes the following roles for administering the whole system rather than just a single database. These roles include but are not limited to *replica set* and *sharded cluster* administrative functions.

clusterAdmin

Provides the greatest cluster-management access. This role combines the privileges granted by the clusterManager (page 88), clusterMonitor (page 89), and hostManager (page 90) roles. Additionally, the role provides the dropDatabase (page 102) action.

clusterManager

Provides management and monitoring actions on the cluster. A user with this role can access the config and local databases, which are used in sharding and replication, respectively.

Provides the following actions on the cluster as a whole:

- addShard (page 101)
- applicationMessage (page 102)
- cleanupOrphaned (page 100)
- flushRouterConfig (page 101)
- listShards (page 101)
- removeShard (page 102)
- replSetConfigure (page 101)
- replSetGetStatus (page 101)
- replSetStateChange (page 101)
- resync (page 101)

Provides the following actions on *all* databases in the cluster:

- enableSharding (page 101)
- moveChunk (page 101)
- splitChunk (page 102)
- splitVector (page 102)

On the config database, provides the following actions on the settings collection:

- insert (page 99)
- remove (page 99)
- update (page 99)

On the `config` database, provides the following actions on all configuration collections and on the `system.indexes`, `system.js`, and `system.namespaces` collections:

- `collStats` (page 103)
- `dbHash` (page 103)
- `dbStats` (page 103)
- `find` (page 99)
- `killCursors` (page 100)

On the `local` database, provides the following actions on the `replset` collection:

- `collStats` (page 103)
- `dbHash` (page 103)
- `dbStats` (page 103)
- `find` (page 99)
- `killCursors` (page 100)

`clusterMonitor`

Provides read-only access to monitoring tools, such as the MongoDB Cloud Manager[88] and Ops Manager[89] monitoring agent.

Provides the following actions on the cluster as a whole:

- `connPoolStats` (page 103)
- `cursorInfo` (page 103)
- `getCmdLineOpts` (page 103)
- `getLog` (page 103)
- `getParameter` (page 102)
- `getShardMap` (page 101)
- `hostInfo` (page 102)
- `inprog` (page 100)
- `listDatabases` (page 103)
- `listShards` (page 101)
- `netstat` (page 103)
- `replSetGetStatus` (page 101)
- `serverStatus` (page 103)
- `shardingState` (page 102)
- `top` (page 104)

Provides the following actions on *all* databases in the cluster:

- `collStats` (page 103)
- `dbStats` (page 103)
- `getShardVersion` (page 101)

[88] https://cloud.mongodb.com/?jmp=docs
[89] https://docs.opsmanager.mongodb.com/current/

Provides the find (page 99) action on all system.profile collections in the cluster.

Provides the following actions on the config database's configuration collections and system.indexes, system.js, and system.namespaces collections:

- collStats (page 103)
- dbHash (page 103)
- dbStats (page 103)
- find (page 99)
- killCursors (page 100)

hostManager

Provides the ability to monitor and manage servers.

Provides the following actions on the cluster as a whole:

- applicationMessage (page 102)
- closeAllDatabases (page 102)
- connPoolSync (page 102)
- cpuProfiler (page 100)
- diagLogging (page 103)
- flushRouterConfig (page 101)
- fsync (page 102)
- invalidateUserCache (page 100)
- killop (page 101)
- logRotate (page 102)
- resync (page 101)
- setParameter (page 103)
- shutdown (page 103)
- touch (page 103)
- unlock (page 100)

Provides the following actions on *all* databases in the cluster:

- killCursors (page 100)
- repairDatabase (page 103)

Backup and Restoration Roles

The admin database includes the following roles for backing up and restoring data:

backup

Provides minimal privileges needed for backing up data. This role provides sufficient privileges to use the MongoDB Cloud Manager[90] backup agent, Ops Manager[91] backup agent, or to use mongodump to back up an entire mongod instance.

[90] https://cloud.mongodb.com/?jmp=docs
[91] https://docs.opsmanager.mongodb.com/current/

Provides the following *actions* (page 99) on the `mms.backup` collection in the `admin` database:

- `insert` (page 99)
- `update` (page 99)

Provides the `listDatabases` (page 103) action on the cluster as a whole.

Provides the `listCollections` (page 103) action on all databases.

Provides the `listIndexes` (page 103) action for all collections.

Provides the `find` (page 99) action on the following:

- all *non*-system collections in the cluster
- all the following system collections in the cluster: `system.indexes`, `system.namespaces`, and `system.js`
- the `admin.system.users` and `admin.system.roles` collections
- legacy `system.users` collections from versions of MongoDB prior to 2.6

To back up the `system.profile` collection, which is created when you activate *database profiling*, you must have **additional** `read` access on this collection. Several roles provide this access, including the `clusterAdmin` (page 88) and `dbAdmin` (page 86) roles.

`restore`

Provides minimal privileges needed for restoring data from backups. This role provides sufficient privileges to use the `mongorestore` tool to restore an entire `mongod` instance.

Provides the following actions on all *non*-system collections and `system.js` collections in the cluster; on the `admin.system.users` and `admin.system.roles` collections in the `admin` database; and on legacy `system.users` collections from versions of MongoDB prior to 2.6:

- `collMod` (page 102)
- `createCollection` (page 99)
- `createIndex` (page 99)
- `dropCollection` (page 100)
- `insert` (page 99)

Provides the `listCollections` (page 103) action on all databases.

Provides the following *additional* actions on `admin.system.users` and legacy `system.users` collections:

- `find` (page 99)
- `remove` (page 99)
- `update` (page 99)

Provides the `find` (page 99) action on all the `system.namespaces` collections in the cluster.

Although, `restore` (page 91) includes the ability to modify the documents in the `admin.system.users` collection using normal modification operations, *only* modify these data using the *user management methods*.

All-Database Roles

The `admin` database provides the following roles that apply to all databases in a `mongod` instance and are roughly equivalent to their single-database equivalents:

readAnyDatabase
Provides the same read-only permissions as `read` (page 85), except it applies to *all* databases in the cluster. The role also provides the `listDatabases` (page 103) action on the cluster as a whole.

readWriteAnyDatabase
Provides the same read and write permissions as `readWrite` (page 86), except it applies to *all* databases in the cluster. The role also provides the `listDatabases` (page 103) action on the cluster as a whole.

userAdminAnyDatabase
Provides the same access to user administration operations as `userAdmin` (page 87), except it applies to *all* databases in the cluster. The role also provides the following actions on the cluster as a whole:

- `authSchemaUpgrade` (page 100)
- `invalidateUserCache` (page 100)
- `listDatabases` (page 103)

The role also provides the following actions on the `admin.system.users` and `admin.system.roles` collections on the `admin` database, and on legacy `system.users` collections from versions of MongoDB prior to 2.6:

- `collStats` (page 103)
- `dbHash` (page 103)
- `dbStats` (page 103)
- `find` (page 99)
- `killCursors` (page 100)
- `planCacheRead` (page 101)

Changed in version 2.6.4: `userAdminAnyDatabase` (page 92) added the following permissions on the `admin.system.users` and `admin.system.roles` collections:

- `createIndex` (page 99)
- `dropIndex` (page 102)

The `userAdminAnyDatabase` (page 92) role does not restrict the permissions that a user can grant. As a result, `userAdminAnyDatabase` (page 92) users can grant themselves privileges in excess of their current privileges and even can grant themselves *all privileges*, even though the role does not explicitly authorize privileges beyond user administration. This role is effectively a MongoDB system *superuser* (page 92).

dbAdminAnyDatabase
Provides the same access to database administration operations as `dbAdmin` (page 86), except it applies to *all* databases in the cluster. The role also provides the `listDatabases` (page 103) action on the cluster as a whole.

Superuser Roles

Several roles provide either indirect or direct system-wide superuser access.

The following roles provide the ability to assign any user any privilege on any database, which means that users with one of these roles can assign *themselves* any privilege on any database:

- `dbOwner` (page 87) role, when scoped to the `admin` database
- `userAdmin` (page 87) role, when scoped to the `admin` database
- `userAdminAnyDatabase` (page 92) role

The following role provides full privileges on all resources:

root

Provides access to the operations and all the resources of the readWriteAnyDatabase (page 92), dbAdminAnyDatabase (page 92), userAdminAnyDatabase (page 92) and clusterAdmin (page 88) roles *combined*.

root (page 93) does **not** include any access to collections that begin with the system. prefix.

For example, without the ability to insert data directly into the:data:*system.users* <*admin.system.users*> and system.roles collections in the admin database. root (page 93) is not suitable for writing or restoring data that have these collections (e.g. with mongorestore.) To perform these kinds of restore operations, provision users with the restore (page 91) role.

Internal Role

__system

MongoDB assigns this role to user objects that represent cluster members, such as replica set members and mongos instances. The role entitles its holder to take any action against any object in the database.

Do not assign this role to user objects representing applications or human administrators, other than in exceptional circumstances.

If you need access to all actions on all resources, for example to run applyOps commands, do not assign this role. Instead, *create a user-defined role* (page 70) that grants anyAction (page 104) on *anyResource* (page 99) and ensure that only the users who need access to these operations have this access.

system.roles Collection

New in version 2.6.

The system.roles collection in the admin database stores the user-defined roles. To create and manage these user-defined roles, MongoDB provides *role management commands*.

system.roles Schema

The documents in the system.roles collection have the following schema:

```
{
  _id: <system-defined id>,
  role: "<role name>",
  db: "<database>",
  privileges:
      [
          {
              resource: { <resource> },
              actions: [ "<action>", ... ]
          },
          ...
      ],
  roles:
      [
          { role: "<role name>", db: "<database>" },
          ...
      ]
}
```

A `system.roles` document has the following fields:

`admin.system.roles.`**`role`**
: The `role` (page 94) field is a string that specifies the name of the role.

`admin.system.roles.`**`db`**
: The `db` (page 94) field is a string that specifies the database to which the role belongs. MongoDB uniquely identifies each role by the pairing of its name (i.e. `role` (page 94)) and its database.

`admin.system.roles.`**`privileges`**
: The `privileges` (page 94) array contains the privilege documents that define the *privileges* (page 15) for the role.

  A privilege document has the following syntax:

  ```
  {
    resource: { <resource> },
    actions: [ "<action>", ... ]
  }
  ```

  Each privilege document has the following fields:

  `admin.system.roles.privileges[n].`**`resource`**
  : A document that specifies the resources upon which the privilege `actions` (page 94) apply. The document has one of the following form:

    ```
    { db: <database>, collection: <collection> }
    ```

    or

    ```
    { cluster : true }
    ```

    See *Resource Document* (page 97) for more details.

  `admin.system.roles.privileges[n].`**`actions`**
  : An array of actions permitted on the resource. For a list of actions, see *Privilege Actions* (page 99).

`admin.system.roles.`**`roles`**
: The `roles` (page 94) array contains role documents that specify the roles from which this role *inherits* (page 15) privileges.

  A role document has the following syntax:

  ```
  { role: "<role name>", db: "<database>" }
  ```

  A role document has the following fields:

  `admin.system.roles.roles[n].`**`role`**
  : The name of the role. A role can be a *built-in role* (page 85) provided by MongoDB or a *user-defined role* (page 15).

  `admin.system.roles.roles[n].`**`db`**
  : The name of the database where the role is defined.

Examples

Consider the following sample documents found in `system.roles` collection of the `admin` database.

A User-Defined Role Specifies Privileges The following is a sample document for a user-defined role `appUser` defined for the `myApp` database:

```
{
  _id: "myApp.appUser",
  role: "appUser",
  db: "myApp",
  privileges: [
        { resource: { db: "myApp" , collection: "" },
          actions: [ "find", "createCollection", "dbStats", "collStats" ] },
        { resource: { db: "myApp", collection: "logs" },
          actions: [ "insert" ] },
        { resource: { db: "myApp", collection: "data" },
          actions: [ "insert", "update", "remove", "compact" ] },
        { resource: { db: "myApp", collection: "system.js" },
          actions: [ "find" ] },
  ],
  roles: []
}
```

The `privileges` array lists the five privileges that the `appUser` role specifies:

- The first privilege permits its actions (`"find"`, `"createCollection"`, `"dbStats"`, `"collStats"`) on all the collections in the `myApp` database *excluding* its system collections. See *Specify a Database as Resource* (page 98).
- The next two privileges permits *additional* actions on specific collections, `logs` and `data`, in the `myApp` database. See *Specify a Collection of a Database as Resource* (page 97).
- The last privilege permits actions on one `system collections` in the `myApp` database. While the first privilege gives database-wide permission for the `find` action, the action does not apply to `myApp`'s system collections. To give access to a system collection, a privilege must explicitly specify the collection. See *Resource Document* (page 97).

As indicated by the empty `roles` array, `appUser` inherits no additional privileges from other roles.

User-Defined Role Inherits from Other Roles The following is a sample document for a user-defined role `appAdmin` defined for the `myApp` database: The document shows that the `appAdmin` role specifies privileges as well as inherits privileges from other roles:

```
{
  _id: "myApp.appAdmin",
  role: "appAdmin",
  db: "myApp",
  privileges: [
                {
                  resource: { db: "myApp", collection: "" },
                  actions: [ "insert", "dbStats", "collStats", "compact", "repairDatabase" ]
                }
              ],
  roles: [
           { role: "appUser", db: "myApp" }
         ]
}
```

The `privileges` array lists the privileges that the `appAdmin` role specifies. This role has a single privilege that permits its actions (`"insert"`, `"dbStats"`, `"collStats"`, `"compact"`, `"repairDatabase"`) on all the collections in the `myApp` database *excluding* its system collections. See *Specify a Database as Resource* (page 98).

The `roles` array lists the roles, identified by the role names and databases, from which the role `appAdmin` inherits privileges.

`system.users` Collection

Changed in version 2.6.

The `system.users` collection in the `admin` database stores user *authentication* (page 6) and *authorization* (page 14) information. To manage data in this collection, MongoDB provides *user management commands*.

`system.users` Schema

The documents in the `system.users` collection have the following schema:

```
{
  _id: <system defined id>,
  user: "<name>",
  db: "<database>",
  credentials: { <authentication credentials> },
  roles: [
           { role: "<role name>", db: "<database>" },
           ...
         ],
  customData: <custom information>
 }
```

Each `system.users` document has the following fields:

`admin.system.users.`**`user`**
: The `user` (page 96) field is a string that identifies the user. A user exists in the context of a single logical database but can have access to other databases through roles specified in the `roles` (page 96) array.

`admin.system.users.`**`db`**
: The `db` (page 96) field specifies the database associated with the user. The user's privileges are not necessarily limited to this database. The user can have privileges in additional databases through the `roles` (page 96) array.

`admin.system.users.`**`credentials`**
: The `credentials` (page 96) field contains the user's authentication information. For users with externally stored authentication credentials, such as users that use *Kerberos* (page 55) or x.509 certificates for authentication, the `system.users` document for that user does not contain the `credentials` (page 96) field.

`admin.system.users.`**`roles`**
: The `roles` (page 96) array contains role documents that specify the roles granted to the user. The array contains both *built-in roles* (page 85) and *user-defined role* (page 15).

  A role document has the following syntax:

  ```
  { role: "<role name>", db: "<database>" }
  ```

  A role document has the following fields:

  `admin.system.users.roles[n].`**`role`**
  : The name of a role. A role can be a *built-in role* (page 85) provided by MongoDB or a *custom user-defined role* (page 15).

  `admin.system.users.roles[n].`**`db`**
  : The name of the database where role is defined.

When specifying a role using the *role management* or *user management* commands, you can specify the role name alone (e.g. `"readWrite"`) if the role that exists on the database on which the command is run.

`admin.system.users.`**`customData`**
The `customData` (page 97) field contains optional custom information about the user.

Example

Changed in version 3.0.0.

Consider the following document in the `system.users` collection:

```
{
  _id : "home.Kari",
  user : "Kari",
  db : "home",
  credentials : {
          "SCRAM-SHA-1" : {
                  "iterationCount" : 10000,
                  "salt" : nkHYXEZTTYmn+hrY994y1Q==",
                  "storedKey" : "wxWGN3ElQ25WbPjACeXdUmN4nNo=",
                  "serverKey" : "h7vBq5tACT/BtrIElY2QTm+pQzM="
          }
  },
  roles : [
             { role: "read", db: "home" },
             { role: "readWrite", db: "test" },
             { role: "appUser", db: "myApp" }
           ],
  customData : { zipCode: "64157" }
}
```

The document shows that a user `Kari` is associated with the `home` database. `Kari` has the `read` (page 85) role in the `home` database, the `readWrite` (page 86) role in the `test` database, and the `appUser` role in the `myApp` database.

Resource Document

The resource document specifies the resources upon which a privilege permits `actions`.

Database and/or Collection Resource

To specify databases and/or collections, use the following syntax:

```
{ db: <database>, collection: <collection> }
```

Specify a Collection of a Database as Resource If the resource document species both the `db` and `collection` fields as non-empty strings, the resource is the specified collection in the specified database. For example, the following document specifies a resource of the `inventory` collection in the `products` database:

```
{ db: "products", collection: "inventory" }
```

For a user-defined role scoped for a non-`admin` database, the resource specification for its privileges must specify the same database as the role. User-defined roles scoped for the `admin` database can specify other databases.

Specify a Database as Resource If only the `collection` field is an empty string (`""`), the resource is the specified database, excluding the `system collections`. For example, the following resource document specifies the resource of the `test` database, excluding the system collections:

```
{ db: "test", collection: "" }
```

For a user-defined role scoped for a non-`admin` database, the resource specification for its privileges must specify the same database as the role. User-defined roles scoped for the `admin` database can specify other databases.

Note: When you specify a database as the resource, system collections are excluded, unless you name them explicitly, as in the following:

```
{ db: "test", collection: "system.js" }
```

System collections include but are not limited to the following:

- `<database>.system.profile`
- `<database>.system.js`
- *system.users Collection* (page 96) in the `admin` database
- *system.roles Collection* (page 93) in the `admin` database

Specify Collections Across Databases as Resource If only the `db` field is an empty string (`""`), the resource is all collections with the specified name across all databases. For example, the following document specifies the resource of all the `accounts` collections across all the databases:

```
{ db: "", collection: "accounts" }
```

For user-defined roles, only roles scoped for the `admin` database can have this resource specification for their privileges.

Specify All Non-System Collections in All Databases If both the `db` and `collection` fields are empty strings (`""`), the resource is all collections, excluding the `system collections`, in all the databases:

```
{ db: "", collection: "" }
```

For user-defined roles, only roles scoped for the `admin` database can have this resource specification for their privileges.

Cluster Resource

To specify the cluster as the resource, use the following syntax:

```
{ cluster : true }
```

Use the `cluster` resource for actions that affect the state of the system rather than act on specific set of databases or collections. Examples of such actions are `shutdown`, `replSetReconfig`, and `addShard`. For example, the following document grants the action `shutdown` on the `cluster`.

```
{ resource: { cluster : true }, actions: [ "shutdown" ] }
```

For user-defined roles, only roles scoped for the `admin` database can have this resource specification for their privileges.

anyResource

The internal resource `anyResource` gives access to every resource in the system and is intended for internal use. **Do not** use this resource, other than in exceptional circumstances. The syntax for this resource is `{ anyResource: true }`.

Privilege Actions

New in version 2.6.

Privilege actions define the operations a user can perform on a *resource* (page 97). A MongoDB *privilege* (page 15) comprises a *resource* (page 97) and the permitted actions. This page lists available actions grouped by common purpose.

MongoDB provides built-in roles with pre-defined pairings of resources and permitted actions. For lists of the actions granted, see *Built-In Roles* (page 85). To define custom roles, see *Create a User-Defined Role* (page 70).

Query and Write Actions

find
: User can perform the `db.collection.find()` method. Apply this action to database or collection resources.

insert
: User can perform the `insert` command. Apply this action to database or collection resources.

remove
: User can perform the `db.collection.remove()` method. Apply this action to database or collection resources.

update
: User can perform the `update` command. Apply this action to database or collection resources.

Database Management Actions

changeCustomData
: User can change the custom information of any user in the given database. Apply this action to database resources.

changeOwnCustomData
: Users can change their own custom information. Apply this action to database resources. See also *Change Your Password and Custom Data* (page 75).

changeOwnPassword
: Users can change their own passwords. Apply this action to database resources. See also *Change Your Password and Custom Data* (page 75).

changePassword
: User can change the password of any user in the given database. Apply this action to database resources.

createCollection
: User can perform the `db.createCollection()` method. Apply this action to database or collection resources.

createIndex
Provides access to the `db.collection.createIndex()` method and the `createIndexes` command. Apply this action to database or collection resources.

createRole
User can create new roles in the given database. Apply this action to database resources.

createUser
User can create new users in the given database. Apply this action to database resources.

dropCollection
User can perform the `db.collection.drop()` method. Apply this action to database or collection resources.

dropRole
User can delete any role from the given database. Apply this action to database resources.

dropUser
User can remove any user from the given database. Apply this action to database resources.

emptycapped
User can perform the `emptycapped` command. Apply this action to database or collection resources.

enableProfiler
User can perform the `db.setProfilingLevel()` method. Apply this action to database resources.

grantRole
User can grant any role in the database to any user from any database in the system. Apply this action to database resources.

killCursors
User can kill cursors on the target collection.

revokeRole
User can remove any role from any user from any database in the system. Apply this action to database resources.

unlock
User can perform the `db.fsyncUnlock()` method. Apply this action to the `cluster` resource.

viewRole
User can view information about any role in the given database. Apply this action to database resources.

viewUser
User can view the information of any user in the given database. Apply this action to database resources.

Deployment Management Actions

authSchemaUpgrade
User can perform the `authSchemaUpgrade` command. Apply this action to the `cluster` resource.

cleanupOrphaned
User can perform the `cleanupOrphaned` command. Apply this action to the `cluster` resource.

cpuProfiler
User can enable and use the CPU profiler. Apply this action to the `cluster` resource.

inprog
User can use the `db.currentOp()` method to return pending and active operations. Apply this action to the `cluster` resource.

invalidateUserCache
Provides access to the `invalidateUserCache` command. Apply this action to the `cluster` resource.

killop
User can perform the `db.killOp()` method. Apply this action to the `cluster` resource.

planCacheRead
User can perform the `planCacheListPlans` and `planCacheListQueryShapes` commands and the `PlanCache.getPlansByQuery()` and `PlanCache.listQueryShapes()` methods. Apply this action to database or collection resources.

planCacheWrite
User can perform the `planCacheClear` command and the `PlanCache.clear()` and `PlanCache.clearPlansByQuery()` methods. Apply this action to database or collection resources.

storageDetails
User can perform the `storageDetails` command. Apply this action to database or collection resources.

Replication Actions

appendOplogNote
User can append notes to the oplog. Apply this action to the `cluster` resource.

replSetConfigure
User can configure a replica set. Apply this action to the `cluster` resource.

replSetGetStatus
User can perform the `replSetGetStatus` command. Apply this action to the `cluster` resource.

replSetHeartbeat
User can perform the `replSetHeartbeat` command. Apply this action to the `cluster` resource.

replSetStateChange
User can change the state of a replica set through the `replSetFreeze`, `replSetMaintenance`, `replSetStepDown`, and `replSetSyncFrom` commands. Apply this action to the `cluster` resource.

resync
User can perform the `resync` command. Apply this action to the `cluster` resource.

Sharding Actions

addShard
User can perform the `addShard` command. Apply this action to the `cluster` resource.

enableSharding
User can enable sharding on a database using the `enableSharding` command and can shard a collection using the `shardCollection` command. Apply this action to database or collection resources.

flushRouterConfig
User can perform the `flushRouterConfig` command. Apply this action to the `cluster` resource.

getShardMap
User can perform the `getShardMap` command. Apply this action to the `cluster` resource.

getShardVersion
User can perform the `getShardVersion` command. Apply this action to database resources.

listShards
User can perform the `listShards` command. Apply this action to the `cluster` resource.

moveChunk
User can perform the `moveChunk` command. In addition, user can perform the `movePrimary` command

provided that the privilege is applied to an appropriate database resource. Apply this action to database or collection resources.

removeShard
User can perform the `removeShard` command. Apply this action to the `cluster` resource.

shardingState
User can perform the `shardingState` command. Apply this action to the `cluster` resource.

splitChunk
User can perform the `splitChunk` command. Apply this action to database or collection resources.

splitVector
User can perform the `splitVector` command. Apply this action to database or collection resources.

Server Administration Actions

applicationMessage
User can perform the `logApplicationMessage` command. Apply this action to the `cluster` resource.

closeAllDatabases
User can perform the `closeAllDatabases` command. Apply this action to the `cluster` resource.

collMod
User can perform the `collMod` command. Apply this action to database or collection resources.

compact
User can perform the `compact` command. Apply this action to database or collection resources.

connPoolSync
User can perform the `connPoolSync` command. Apply this action to the `cluster` resource.

convertToCapped
User can perform the `convertToCapped` command. Apply this action to database or collection resources.

dropDatabase
User can perform the `dropDatabase` command. Apply this action to database resources.

dropIndex
User can perform the `dropIndexes` command. Apply this action to database or collection resources.

fsync
User can perform the `fsync` command. Apply this action to the `cluster` resource.

getParameter
User can perform the `getParameter` command. Apply this action to the `cluster` resource.

hostInfo
Provides information about the server the MongoDB instance runs on. Apply this action to the `cluster` resource.

logRotate
User can perform the `logRotate` command. Apply this action to the `cluster` resource.

reIndex
User can perform the `reIndex` command. Apply this action to database or collection resources.

renameCollectionSameDB
Allows the user to rename collections on the current database using the `renameCollection` command. Apply this action to database resources.

Additionally, the user must either *have* `find` (page 99) on the source collection or *not have* `find` (page 99) on the destination collection.

If a collection with the new name already exists, the user must also have the `dropCollection` (page 100) action on the destination collection.

repairDatabase
User can perform the `repairDatabase` command. Apply this action to database resources.

setParameter
User can perform the `setParameter` command. Apply this action to the `cluster` resource.

shutdown
User can perform the `shutdown` command. Apply this action to the `cluster` resource.

touch
User can perform the `touch` command. Apply this action to the `cluster` resource.

Diagnostic Actions

collStats
User can perform the `collStats` command. Apply this action to database or collection resources.

connPoolStats
User can perform the `connPoolStats` and `shardConnPoolStats` commands. Apply this action to the `cluster` resource.

cursorInfo
User can perform the `cursorInfo` command. Apply this action to the `cluster` resource.

dbHash
User can perform the `dbHash` command. Apply this action to database or collection resources.

dbStats
User can perform the `dbStats` command. Apply this action to database resources.

diagLogging
User can perform the `diagLogging` command. Apply this action to the `cluster` resource.

getCmdLineOpts
User can perform the `getCmdLineOpts` command. Apply this action to the `cluster` resource.

getLog
User can perform the `getLog` command. Apply this action to the `cluster` resource.

indexStats
User can perform the `indexStats` command. Apply this action to database or collection resources.

Changed in version 3.0: MongoDB 3.0 removes the `indexStats` command.

listDatabases
User can perform the `listDatabases` command. Apply this action to the `cluster` resource.

listCollections
User can perform the `listCollections` command. Apply this action to database resources.

listIndexes
User can perform the `ListIndexes` command. Apply this action to database or collection resources.

netstat
User can perform the `netstat` command. Apply this action to the `cluster` resource.

serverStatus
User can perform the `serverStatus` command. Apply this action to the `cluster` resource.

validate
User can perform the `validate` command. Apply this action to database or collection resources.

top
User can perform the `top` command. Apply this action to the `cluster` resource.

Internal Actions

anyAction
Allows any action on a resource. **Do not** assign this action except for exceptional circumstances.

internal
Allows internal actions. **Do not** assign this action except for exceptional circumstances.

Default MongoDB Port

The following table lists the default TCP ports used by MongoDB:

Default Port	Description
`27017`	The default port for `mongod` and `mongos` instances. You can change this port with `port` or *`--port`*.
`27018`	The default port when running with *`--shardsvr`* runtime operation or the `shardsvr` value for the `clusterRole` setting in a configuration file.
`27019`	The default port when running with *`--configsvr`* runtime operation or the `configsvr` value for the `clusterRole` setting in a configuration file.
`28017`	The default port for the web status page. The web status page is always accessible at a port number that is `1000` greater than the port determined by `port`.

System Event Audit Messages

Note: Available only in MongoDB Enterprise[92].

Audit Message

The *event auditing feature* (page 16) can record events in JSON format. To configure auditing output, see *Configure System Events Auditing* (page 78)

The recorded JSON messages have the following syntax:

```
{
  atype: <String>,
  ts : { "$date": <timestamp> },
  local: { ip: <String>, port: <int> },
  remote: { ip: <String>, port: <int> },
  users : [ { user: <String>, db: <String> }, ... ],
  roles: [ { role: <String>, db: <String> }, ... ],
  param: <document>,
```

[92] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs

```
  result: <int>
}
```

field String atype Action type. See *Audit Event Actions, Details, and Results* (page 105).

field document ts Document that contains the date and UTC time of the event, in ISO 8601 format.

field document local Document that contains the local `ip` address and the `port` number of the running instance.

field document remote Document that contains the remote `ip` address and the `port` number of the incoming connection associated with the event.

field array users Array of user identification documents. Because MongoDB allows a session to log in with different user per database, this array can have more than one user. Each document contains a `user` field for the username and a `db` field for the authentication database for that user.

field array roles Array of documents that specify the *roles* (page 14) granted to the user. Each document contains a `role` field for the name of the role and a `db` field for the database associated with the role.

field document param Specific details for the event. See *Audit Event Actions, Details, and Results* (page 105).

field integer result Error code. See *Audit Event Actions, Details, and Results* (page 105).

Audit Event Actions, Details, and Results

The following table lists for each `atype` or action type, the associated `param` details and the `result` values, if any.

atype	param	result
`authenticate`	`{ user: <user name>, db: <database>, mechanism: <mechanism> }`	`0` - Success `18` - Authentication Failed
`authCheck`	`{ command: <name>, ns: <database>.<collection>, args: <command object> }` `ns` field is optional. `args` field may be redacted.	`0` - Success `13` - Unauthorized to perform the operation. By default, the auditing system logs only the authorization failures. To enable the system to log authorization successes, use the `auditAuthorizationSuccess` parameter. [93]
`createCollection` (page 99)	`{ ns: <database>.<collection> }`	`0` - Success
`createDatabase`	`{ ns: <database> }`	`0` - Success

Continued on next page

[93] Enabling `auditAuthorizationSuccess` degrades performance more than logging only the authorization failures.

Table 1 – continued from previous page

atype	param	result
createIndex (page 99)	{ ns: <database>.<collection>, indexName: <index name>, indexSpec: <index specification> }	0 - Success
renameCollection	{ old: <database>.<collection>, new: <database>.<collection> }	0 - Success
dropCollection (page 100)	{ ns: <database>.<collection> }	0 - Success
dropDatabase (page 102)	{ ns: <database> }	0 - Success
dropIndex (page 102)	{ ns: <database>.<collection>, indexName: <index name> }	0 - Success
createUser (page 100)	{ user: <user name>, db: <database>, customData: <document>, roles: [{ role: <role name>, db: <database> }, ...] } The customData field is optional.	0 - Success
dropUser (page 100)	{ user: <user name>, db: <database> }	0 - Success
dropAllUsersFromDatabase	{ db: <database> }	0 - Success

Continued on next page

Table 1 – continued from previous page

atype	param	result
updateUser	{ user: <user name>, db: <database>, passwordChanged: <boolean>, customData: <document>, roles: [{ role: <role name>, db: <database> }, ...] } The customData field is optional.	0 - Success
grantRolesToUser	{ user: <user name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...] }	0 - Success
revokeRolesFromUser	{ user: <user name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...] }	0 - Success

Continued on next page

Table 1 – continued from previous page

atype	param	result

createRole (page 100)	{
	role: <role name>,
	db: <database>,
	roles: [
	{
	role: <role name>,
	db: <database>
	},
	...
	],
	privileges: [
	{
	resource: <resource document>,
	actions: [<action>, ...]
	},
	...
	]
	}
	The roles and the privileges fields are optional.
	For details on the resource document, see *Resource Document* (page 97).
	For a list of actions, see *Privilege Actions* (page 99).
		0 - Success

updateRole	{
	role: <role name>,
	db: <database>,
	roles: [
	{
	role: <role name>,
	db: <database>
	},
	...
	],
	privileges: [
	{
	resource: <resource document>,
	actions: [<action>, ...]
	},
	...
	]
	}
	The roles and the privileges fields are optional.
	For details on the resource document, see *Resource Document* (page 97).
	For a list of actions, see *Privilege Actions* (page 99).
		0 - Success

Continued on next page

Table 1 – continued from previous page

atype	param	result
dropRole (page 100)	{ role: <role name>, db: <database> }	0 - Success
dropAllRolesFromDatabase	{ db: <database> }	0 - Success
grantRolesToRole	{ role: <role name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...] }	0 - Success
revokeRolesFromRole	{ role: <role name>, db: <database>, roles: [{ role: <role name>, db: <database> }, ...] }	0 - Success
grantPrivilegesToRole	{ role: <role name>, db: <database>, privileges: [{ resource: <resource document>, actions: [<action>, ...] }, ...] } For details on the resource document, see *Resource Document* (page 97). For a list of actions, see *Privilege Actions* (page 99).	0 - Success

Continued on next page

Table 1 – continued from previous page

atype	param	result
revokePrivilegesFromRole	{ role: <role name>, db: <database name>, privileges: [{ resource: <resource document>, actions: [<action>, ...] }, ...] } For details on the resource document, see *Resource Document* (page 97). For a list of actions, see *Privilege Actions* (page 99).	0 - Success
replSetReconfig	{ old: <configuration>, new: <configuration> } Indicates membership change in the replica set. The old field is optional.	0 - Success
enableSharding (page 101)	{ ns: <database> }	0 - Success
shardCollection	{ ns: <database>.<collection>, key: <shard key pattern>, options: { unique: <boolean> } }	0 - Success
addShard (page 101)	{ shard: <shard name>, connectionString: <hostname>:<port>, maxSize: <maxSize> } When a shard is a replica set, the connectionString includes the replica set name and can include other members of the replica set.	0 - Success
removeShard (page 102)	{ shard: <shard name> }	0 - Success
shutdown (page 103)	{ } Indicates commencement of database shutdown.	0 - Success

Continued on next page

Table 1 – continued from previous page

atype	param	result
applicationMessage (page 102)	{ msg: <custom message string> } See logApplicationMessage.	0 - Success

4.3 Security Release Notes Alerts

Security Release Notes **(page 111)** Security vulnerability for password.

Security Release Notes

Access to system.users Collection

Changed in version 2.4.

In 2.4, only users with the userAdmin role have access to the system.users collection.

In version 2.2 and earlier, the read-write users of a database all have access to the system.users collection, which contains the user names and user password hashes. [94]

Password Hashing Insecurity

If a user has the same password for multiple databases, the hash will be the same. A malicious user could exploit this to gain access on a second database using a different user's credentials.

As a result, always use unique username and password combinations for each database.

Thanks to Will Urbanski, from Dell SecureWorks, for identifying this issue.

5 Security Checklist

This documents provides a list of security measures that you should implement to protect your MongoDB installation.

5.1 Require Authentication

Enable MongoDB authentication and specify the authentication mechanism. You can use the MongoDB authentication mechanism or an existing external framework. Authentication requires that all clients and servers provide valid credentials before they can connect to the system. In clustered deployments, enable authentication for each MongoDB server.

See *Authentication* (page 6), *Enable Client Access Control* (page 40), and *Enable Authentication in a Sharded Cluster* (page 41).

[94] Read-only users do not have access to the system.users collection.

5.2 Configure Role-Based Access Control

Create a user administrator **first**, then create additional users. Create a unique MongoDB user for each person and application that accesses the system.

Create roles that define the exact access a set of users needs. Follow a principle of least privilege. Then create users and assign them only the roles they need to perform their operations. A user can be a person or a client application.

See *Authorization* (page 14), *Create a User Administrator* (page 67), and *Manage User and Roles* (page 69), .

5.3 Encrypt Communication

Configure MongoDB to use TLS/SSL for all incoming and outgoing connections. Use TLS/SSL to encrypt communication between `mongod` and `mongos` components of a MongoDB client as well as between all applications and MongoDB.

See *Configure mongod and mongos for TLS/SSL* (page 26).

5.4 Limit Network Exposure

Ensure that MongoDB runs in a trusted network environment and limit the interfaces on which MongoDB instances listen for incoming connections. Allow only trusted clients to access the network interfaces and ports on which MongoDB instances are available.

See the `bindIp` setting, and see *Configure Linux iptables Firewall for MongoDB* (page 20) and *Configure Windows netsh Firewall for MongoDB* (page 23).

5.5 Audit System Activity

Track access and changes to database configurations and data. MongoDB Enterprise[95] includes a system auditing facility that can record system events (e.g. user operations, connection events) on a MongoDB instance. These audit records permit forensic analysis and allow administrators to verify proper controls.

See *Auditing* (page 16) and *Configure System Events Auditing* (page 78).

5.6 Encrypt and Protect Data

Encrypt MongoDB data on each host using file-system, device, or physical encryption. Protect MongoDB data using file-system permissions. MongoDB data includes data files, configuration files, auditing logs, and key files.

5.7 Run MongoDB with a Dedicated User

Run MongoDB processes with a dedicated operating system user account. Ensure that the account has permissions to access data but no unnecessary permissions.

See `http://docs.mongodb.org/manual/installation` for more information on running MongoDB.

[95] http://www.mongodb.com/products/mongodb-enterprise?jmp=docs

5.8 Run MongoDB with Secure Configuration Options

MongoDB supports the execution of JavaScript code for certain server-side operations: `mapReduce`, `group`, and `$where`. If you do not use these operations, disable server-side scripting by using the *`--noscripting`* option on the command line.

Use only the MongoDB wire protocol on production deployments. Do **not** enable the following, all of which enable the web server interface: `enabled`, `net.http.JSONPEnabled`, and `net.http.RESTInterfaceEnabled`. Leave these *disabled*, unless required for backwards compatibility.

Keep input validation enabled. MongoDB enables input validation by default through the `wireObjectCheck` setting. This ensures that all documents stored by the `mongod` instance are valid *BSON*.

5.9 Request a Security Technical Implementation Guide (where applicable)

The Security Technical Implementation Guide (STIG) contains security guidelines for deployments within the United States Department of Defense. MongoDB Inc. provides its STIG, upon request, for situations where it is required. Please request a copy[96] for more information.

5.10 Consider Security Standards Compliance

For applications requiring HIPAA or PCI-DSS compliance, please refer to the MongoDB Security Reference Architecture[97] to learn more about how you can use the key security capabilities to build compliant application infrastructure.

[96] http://www.mongodb.com/lp/contact/stig-requests
[97] http://info.mongodb.com/rs/mongodb/images/MongoDB_Security_Architecture_WP.pdf