An Analysis and Overview of MongoDB Security

Matthew Trudeau
University of Rochester
Department of Data Science
mtrudeau@ur.rochester.edu

Joshua Kolodny
University of Rochester
Department of Data Science
jkolodn2@ur.rochester.edu

Abstract

NoSQL databases are becoming increasingly popular as the scale of data dealt with has grown tremendously. As NoSQL database technologies increase in popularity, their probability of being a target of hacking increases as well. Unfortunately, NoSQL databases are susceptible to many of the same hacking methods that SQL databases are vulnerable to such as injections. In this paper, we focus our attention on MongoDB, a document-based NoSQL database. MongoDB has a number of security features built-in including authorization, authentication, and TLS/SSL encryption. Failure to utilize these security features in a MongoDB deployment can result in major security risks as demonstrated by the data ransoms in January 2017. These security features will be analyzed in terms of effectiveness in fighting off an attack and restrictions on database performance. Additionally, this paper will detail pitfalls of MongoDB security and future security measures that could be added to the current open-source framework.

1. Introduction

The popularity of relational databases over the years has led to a problem described as big data. Since relational databases fail to scale well on large datasets, companies have turned to alternative data storage methods such as NoSQL. NoSQL databases were designed almost exclusively for speed and scalability. The rise of NoSQL has allowed companies like Google and Facebook to ensure efficiency of their systems in the big data era.

Unfortunately, this need for scalability and speed has resulted in security being an afterthought, with many NoSQL technologies susceptible to hacking. Specifically, the focus of this paper turns to the open source NoSQL document oriented database MongoDB. Before implementing a database like MongoDB, companies must be aware of the security risks involved. With the amount of data being collected now higher than ever, security and privacy of this data is absolutely vital. If a company suffers a data breach, the result is ostensibly losing the trust of customers which would inevitably lead to decreased profits.

Over the years, MongoDB has added numerous built-in security features. However, it is the responsibility of the database administrator to take advantage of what MongoDB provides. Using these security features will greatly increase the likelihood that data is protected.

2. Security Failures

2.1 NoSQL

Like relational databases, NoSQL database technologies are susceptible to hacking, injection, and deletion based on security failures, perhaps even more so. As NoSQL technologies were first being developed and used, they lacked adequate security functionality.[1] While security measures have improved recently, NoSQL database technologies are still susceptible to many of the same problems mentioned. Additionally, as NoSQL is a blanket phrase used to refer to various different database technologies, the problems that NoSQLs face can be unique and varied.

2.2 MongoDB

MongoDB's notable security problems are "lack of encryption support for data files, weak authentication between clients and servers, simple authentication, and vulnerability to SQL injection and DOS attack."[2]

One of the problems mentioned, of weak authentication between clients and servers, was responsible for a data breach in early 2017 which saw roughly 30,000 MongoDB instances exposed. While the issue that caused this data breach is by no means intractable, and was due to a default setting not being changed by users, there are other issues that seem more difficult to deal with.

For example, MongoDB databases are susceptible to both JavaScript Injection and HTTP trespassing. HTTP trespassing can be executed by making specific alterations to the source code of a website downloaded by a browser. This can be accomplished if a hacker has access to information about the MongoDB database: the database name, collection name, port number, username, and password.[3] The JavaScript Injections are accomplished by hiding JavaScript source code within a MongoDB query statement. This can result in the query appearing to run once, but actually running several times.

3. MongoDB Security Features

This section introduces the built-in MongoDB (version 3.4) security features that are used to prevent popular database attacks: authentication, authorization, encryption, auditing, network exposure, and injection prevention. As noted below, not all of these features are completely effective. At the very least, many result in decreased database speed.

3.1 Authentication

Authentication of a user is vital in any database implementation. Determining who the user is allows for authorization and other security measures to be applied appropriately. MongoDB uses SCRAM-SHA-1 as the default method for user authentication. The Internet Engineering Task Force (IETF) established SCRAM-SHA-1 to formally define how to securely implement a challenge-response mechanism that authenticates users with a password.[6]

MongoDB previously defaulted to authorizing users with MongoDB Challenge and Response (MongoDB-CR). SCRAM-SHA-1 has a number of advantages over MongoDB-CR such as a tunable work factor, per-user random salts, a stronger hash function (SHA-1 rather than MD5), and bidirectional authentication of the server and client. It is absolutely vital that MongoDB implementations of versions before 3.0 update to the latest version to get these upgrades. Specifically, we will look closer at the hash functions SHA-1 and MD5.

MD5 has been known to suffer from vulnerabilities since 1996.[5] Over the years, even more security flaws of MD5 have been found including the potential for collisions. While MD5 is a faster hash function than SHA-1, SHA-1 is widely accepted as being more secure.

However, it is important to understand that SHA-1 is still vulnerable to attacks. In 2005, SHA-1 was also found to suffer from collisions and will not survive attacks from well funded opponents.[5] SHA-1 has since been taken over by the prefered SHA-2 and SHA-3 cryptographic functions. There are open source libraries to help programmers apply SHA-3 to their MongoDB implementation. However, it is not built-in.

Choosing the best built-in MongoDB authentication method really depends on the specific situation. For large-scale organizational use of MongoDB, investing in the MongoDB Enterprise edition seems to be a reasonable option. The enterprise edition of MongoDB unlocks more authentication methods such as Kerberos Authentication, and LDAP Proxy Authentication.

3.2 Authorization

Authentication is a prerequisite for authorization. Now that unique instances of users can be reliably identified, each user can be assigned predefined roles. Roles are used to grant users access to different MongoDB resources.[3]

Roles can be defined in the admin database and describe what privileges all users have over certain databases and collections. Roles can inherit privileges from other roles to expand on legal user actions. Database administrators have the responsibility of creating new users and assigning them roles. Administrators have the power of using MongoDB built in roles or can create roles for a specific purpose.

It is vital that database administrators take full advantage of assigning roles. Limiting user behavior will limit the danger occurring from a single account being hacked. A hacked account only presents a disastrous outcome if that user is a database administrator.

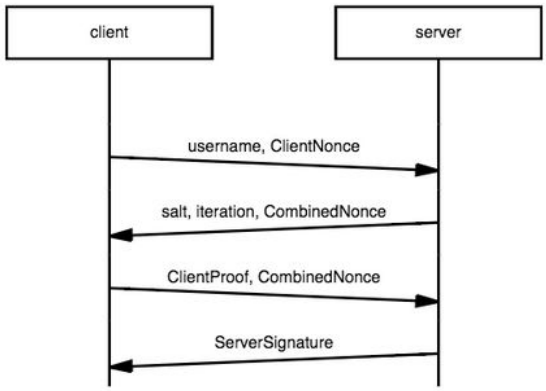

Fig. 1: A visualization of the exchange of messages during an authentication session.[7]

3.3 Encryption

MongoDB supports two different kinds of encryption. By default, it uses AES256-CBC, which is the Advanced Encryption Standard running in Cipher Block Chaining mode. Additionally, MongoDB supports AES256-GCM, which is known as Galois/Counter Mode. Two different types of keys are used in the data encryption process: master keys and database keys. The data within the database is encrypted using the database keys, and the database keys are in turn encrypted with the master key.

MongoDB does not offer any in-house features for application level encryption. To encrypt each field or document, MongoDB documentation suggests writing a custom encryption/decryption methods or using solutions created by one of their partners.

MongoDB also supports transport encryption, such as TLS/SSL, to encrypt network traffic. The implementation of TLS/SSL makes use of OpenSSL libraries, only using SSL ciphers that use a key that is at least 128-bit in length.

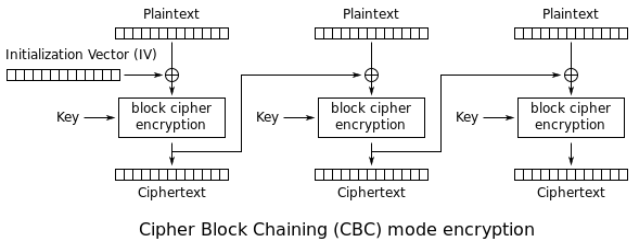

Fig. 2: A conceptual model of how AES-CBC operates.[8]

3.4 Auditing

Auditing is arguably the most important piece of security because it allows database administrators to track the history of the database. An auditing feature built-in to MongoDB enterprise allows administrators to log queries made against the database. Tracking system activity in this way is useful for a variety of reasons, such as identifying malicious attacks due to multiple failed authentication attempts.

The traffic of a database can vary greatly. To enhance the readability of these logs, MongoDB can be configured to filter events and write them to different outputs. These events can be written to the console, the syslog, a JSON file, or a BSON file.

Some operations that the MongoDB auditing feature will log includes schema changes (DDL), authentication attempts, authorization changes, and CRUD (create, read, update, delete) operations. The data logged for a certain event can be customized and can include the action being performed, the parameters sent, a timestamp, the user, the user role(s), and more.

By default, MongoDB does ignore some operations completely and does not log them. It is recommended, however, to use the auditing system to log every event.

The auditing system writes events to an in-memory buffer and periodically writes this buffer to disk. During a single connection, MongoDB promises that if an event has been logged then this means all previous events in that connection has been logged.

In case of a hard shutdown, special treatment of DDL events are necessary to maintain a reliable database history. To maintain a durable state of the database, DDL events cause all events in that connection to be immediately written to disk.

3.5 Network Exposure

MongoDB should be implemented such that network exposure is limited. This is done by running MongoDB on a trusted network and only allowing trusted clients to interface with the network.

By default, MongoDB follows the best practice of limiting network access to localhost. In application, this implementation is not common since databases tend to be accessed remotely. However, this is an important starting point because authentication, authorization, and other security measures should be established prior to making the MongoDB instance available to the public.

Once MongoDB is being hosted on a network and listening for connection attempts, it is vital that only trusted connections to that network are allowed. Use of firewalls and VPN can limit network traffic to only trusted users. It is also recommended that the port used to listen for connections is changed from the default, which is 27017. Automated attacks crawl through networks attempting to connect to MongoDB deployments using the default port.

Even if a malicious user accesses the network, authentication and authorization security implementations should prevent an attacker from completing a disastrous attack such as a data ransom.[3]

3.6 Injection Prevention

Using injection methods as a means of hacking is possible in MongoDB. As mentioned in the security failures section, injections can occur in MongoDB. However, not all types of injections are possible due built-in security features.

One example of this is the fact that SQL injections are not possible. This is because, when queries are assembled in MongoDB, they build BSON objects instead of a string. Nevertheless, injections are a still possible—both in the form of HTTP trespassing and JavaScript Injections.

The MongoDB documentation presents ways in which these injections can be avoided. First, by being mindful of which MongoDB operations allow for the running of arbitrary JavaScript expressions: $where, mapReduce, and group; second, by using the “CodeWScope” mechanism if user-supplied values must be passed a $where clause.

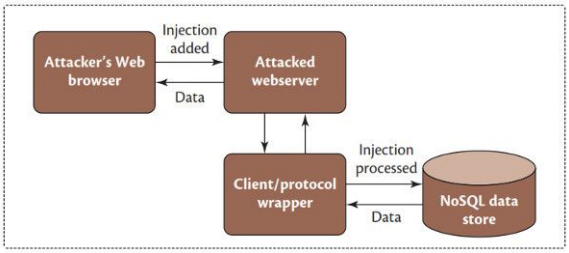

Fig. 3: A schematic showing how the process of injection hacking proceeds.[9]

4. Future Improvements

Of the built-in security features detailed in this paper, there are clear issues with authentication and encryption.

The MongoDB authentication method defaults to a relatively costly hashing algorithm, SHA-1, which is proven to break under certain conditions. Implementing SHA-3 or paying for the enterprise edition are possible options to fix this. However, it seems evident that the primary security risk for MongoDB is that database administrators do not configure database security appropriately if at all.

Defaulting to an effective, free, authentication method is necessary if MongoDB wants to maintain its popularity. Otherwise, hacking user accounts will eventually become commonplace for the free deployment. Database administrators will inevitably seek alternative data storage options.

Finally, application level encryption must be implemented independent of MongoDB instances. To avoid interception of data, all fields must be encrypted at every step. As mentioned previously, database security best practices are commonly ignored or left until the end. Application level encryption should be built-in to encourage programmers to easily take advantage of the feature.

As one of the most popular NoSQL database management systems available, MongoDB should default to database security best practices. Of course, it is still up to the discretion of the database administrator to implement built-in features. Please note, implementing all built-in security features is a must for any successful database.

5. Conclusion

It would seem that MongoDB's built in features provide adequate security, especially for deployments running on the enterprise edition. However, there are still two main issues that need to be addressed: first, the use of SHA-1, which has been known to be vulnerable to attacks for over a decade; second, the absence of built-in application level encryption. These issues notwithstanding, MongoDB (especially its enterprise edition) seems to be a secure and attractive option for big data needs.

The hacks that occured in early 2017 were due to MongoDB's questionable selection of default settings, and also due to users not following best practices. With that in mind, MongoDB users should be motivated to make use of the security features offered, and to ensure that a database's settings are set up appropriately with respect to their security needs.

However, a better way to ensure compliance with the best security practices of MongoDB would be to have the security features as "opt-out" instead of "opt-in." If all security features were on by default, speed may suffer, but the security of general deployments would improve.

Additionally, DBAs may need to turn off these default settings to improve performance. As a result, they would likely become familiar with all of the security features available to them. Alternatively, as the default settings are now, DBAs have no performance-related incentive to learn about the built-in security features.

6. References

[1] Hou, Boyu, et al. "Towards Analyzing MongoDB NoSQL Security and Designing Injection Defense Solution." *2017 IEEE 3rd International Conference on Big Data Security on Cloud*, July 2017.

[2] Sahafizadeh, Ebrahim, and Mohammad Nematbakhsh. "A Survey on Security Issues in Big Data and NoSQL." *Advances in Computer Science: an International Journal*, vol. 4, no. 4, July 2015, pp. 68–72.

[3] "Security." *Security — MongoDB Manual 3.4*, MongoDB, Inc, docs.mongodb.com/manual/security/.

[4] Al-Ithawi, O. *A Security Comparison between MySQL and MongoDB*. n.d. TS.

[5] Stevens, Marc, et al. "SHAttered." *SHAttered*, Google Research, shattered.it/.

[6] "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms." *IETF Tools*, tools.ietf.org/html/rfc5802.

[7] "Improved Password-Based Authentication in MongoDB 3.0: SCRAM Explained - Pt. 1." *MongoDB*, www.mongodb.com/blog/post/improved-password-based-authentication-mongodb-30-scram-explained-part-1.

[8] "Block cipher mode of operation." *Wikipedia*, Wikimedia Foundation, 23 Nov. 2017, en.wikipedia.org/wiki/Block_cipher_mode_of_operation.

[9]"Analysis and Mitigation of NoSQL Injections." *InfoQ*, www.infoq.com/articles/nosql-injections-analysis.