A Web Based Search Engine

1st Prasun Kumar
CSE
Parul University
vadodara, india

2nd Singh AbhishekKumar
CSE
Parul University
vadodara, india

3rd Manish Kumar
CSE
Parul University
vadodara, india

4th Shyam Tiwari
CSE
Parul University
vadodara, india

5th Vaibhavi Jani
CSE,PIET
Parul University
vadodara, india

***Abstract*—Competitive programming involves solving algorithm-mic problems from various online platforms such as Codeforces, Leetcode, HackerRank, etc. These platforms offer a wide range of problems with varying difficulty levels and categories. However, finding relevant and challenging problems from these platforms can be time-consuming and overwhelming. In addition, many competitive programmers may not be familiar with all the platforms and may not know where to start looking for problems. To address this issue, we propose the development of a web-based search engine that aggregates problems from different platforms and allows users to search for specific problems based on various criteria such as difficulty level, tags, and contest name. The proposed search engine will use web scraping techniques to extract problem data from different platforms. The scraped data will be stored in a database and indexed using a search engine such as Elasticsearch. The search engine will provide users with a simple and intuitive interface to search for problems based on different criteria. Users will also be able to filter the results based on their preferences. Developing a web-based search engine for competitive programming problems from platforms like codeforces, Leetcode, etc, and writing the code for scraping using Nodejs for taking data (IDs, URL links, and tags of a question) from websites like codeforces.**

Keywords: Competitive programming involves solving algorithmic problems from various online platforms like Code-forces, Leetcode, HackerRank, etc.

I. Introduction

Developing a web-based search engine for competitive programming problems. The search engine will aggregate problems from different platforms and allow users to search for specific problems based on various criteria such as difficulty level, tags, and contest name. The proposed solution involves web scraping techniques to extract problem data from different platforms, storing the scraped data in a database, and indexing it using a search engine like Elasticsearch. The search engine will provide users with a simple and intuitive interface to search for problems based on different criteria.

Competitive programming has gained immense popularity among developers and computer science enthusiasts as a means of honing their problem-solving skills and algorithmic understanding. However, navigating the plethora of problems scattered across various online platforms can be a daunting task for both beginners and experienced programmers alike. To address this challenge, we propose the development of a comprehensive web-based search engine tailored specifically for competitive programming problems. This search engine aims to streamline the process of discovering relevant problems by aggregating data from multiple platforms and providing users with intuitive search functionalities.

To ensure optimal performance and user experience, the scraped problem data will be stored and indexed using advanced indexing technologies like Elasticsearch. This indexing process will facilitate swift and accurate retrieval of problems based on various search criteria, including difficulty level, problem tags, contest names, and more. By providing a seamless interface that empowers users to refine their search queries and explore relevant problems effortlessly, our search engine aims to democratize access to high-quality competitive programming challenges and foster a vibrant community of aspiring programmers striving for mastery in algorithmic problem-solving.

II. APPLICATION

A. Education

The platform can be used in educational institutions to help students and teachers find relevant and challenging problems to improve their algorithmic problem-solving skills. The platform can also be used to evaluate the progress of students and provide feedback.

B. Recruitment

Companies that require algorithmic problem-solving skills as part of their recruitment process can use the platform to evaluate candidates. The platform can be used to provide a set of problems that are relevant to the position being offered, and

candidates can be evaluated based on their solutions to these problems.

C. Self-Learning

The platform can be used by individuals who want to improve their algorithmic problem-solving skills for personal or professional development. The platform provides a means of finding relevant and challenging problems, allowing individuals to track their progress and evaluate their skills.

D. Competitive Programming

The platform can be used by competitive programmers to find problems for practicing and improving their skills. The platform can also be used to participate in online coding competitions and hackathons.

III. 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.

A. 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 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 preferred 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.

B. 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 to use MongoDB built-in roles or can create roles for a specific purpose.

Database administrators must 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.

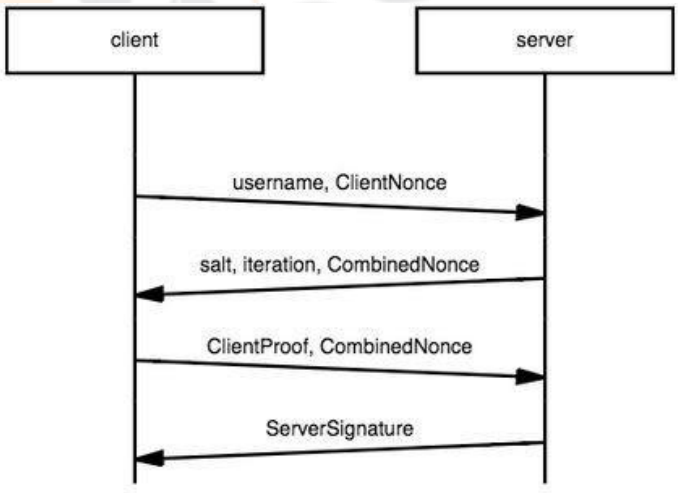

Fig. 1. A visualization of the exchange of messages during an authentication session.

C. 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.

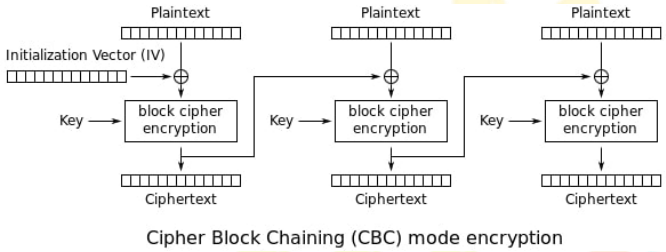

Fig. 2. A conceptual model of how AES-CBC operates.

D. 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 before making the MongoDB instance available to the public.

Once MongoDB is being hosted on a network and listening for connection attempts, only trusted connections to that network must be allowed. The 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 be 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.

E. 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 to 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, build BSON objects instead of a string. Nevertheless, injections are 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, map Reduce, and group; second, by using the "CodeScope" mechanism if user-supplied values must be passed a where clause.

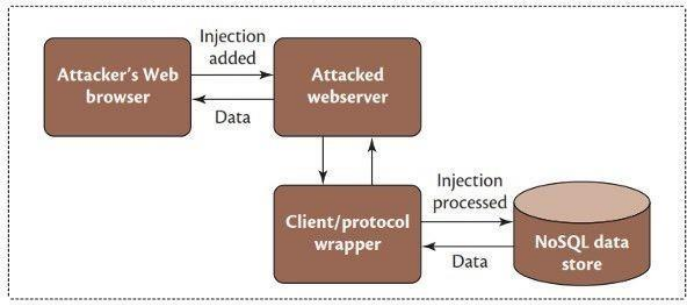

Fig. 3. A schematic showing how the process of injection hacking proceeds.

IV. 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 the primary security risk for MongoDB is that the 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 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 No-SQL 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 are a must for any successful database.

V. 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 occurred 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 for 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.

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-a uthentication-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] Chodorow, K. (2013). MongoDB: The Definitive Guide (2nd ed.). O'Reilly Media.

[10] Pospíšil, J., Pokorný, J. (2016). Security Analysis of NoSQL Databases. Information Sciences, 367-368, 714-729. doi:10.1016/j.ins.2016.06.028

[11] Nayak, A., Joshi, V., Kumar, V. (2018). A Comparative Analysis of Security Features in SQL and NoSQL Databases. International Journal of Advanced Computer Science and Applications, 9(9), 144-150. doi:10.14569/ijacsa.2018.090920

[12] da Silva, F. A., de Oliveira, J. P. M. (2017). A Survey on Security Issues in NoSQL Databases. In 2017 IEEE International Conference on Information Reuse and Integration (IRI) (pp. 435-440). IEEE. doi:10.1109/IRI.2017.105

[13] Javed, M. A., Agarwal, S. (2015). Securing NoSQL Database: A Study and Evaluation. In 2015 International Conference on Green Computing and Internet of Things (ICGCIoT) (pp. 946-951). IEEE. doi:10.1109/ICGCIoT.2015.7380571

[14] Lin, Y., Zhou, L., Zhang, X., Jin, H. (2019). A Study of MongoDB Security Strategies. Journal of Physics: Conference Series, 1168(1), 012079. doi:10.1088/1742-6596/1168/1/012079

[15] Sosík, P., Němcová, A. (2015). Security Issues of NoSQL Databases: A Survey. In 2015 38th International Conference on Telecommunications and Signal Processing (TSP) (pp. 142-146). IEEE. doi:10.1109/TSP.2015.7296301

[16] Hasan, R., Rahman, M., Chowdhury, M. S. (2016). NoSQL Database: Security Threats and Countermeasures. Journal of King Saud University - Computer and Information Sciences, 28(3), 335-345. doi:10.1016/j.jksuci.2014.09.003

[17] Sanzgiri, U., Karmarkar, A., Takate, A. (2014). Security analysis of NoSQL databases. In 2014 International Conference on Contemporary Computing and Informatics (IC3I) (pp. 156-161). IEEE. doi:10.1109/IC3I.2014.7019803

[18] Bhowmick, S., Thakur, P. (2017). A Survey on Security Issues and Solutions in NoSQL Databases. International Journal of Computer Applications, 171(11), 32-37. doi:10.5120/ijca2017915327

[19] Jindal, S., Singh, M. (2019). A Comparative Study of Security Features in SQL and NoSQL Databases. In 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-6). IEEE. doi:10.1109/ICECCT.2019.8869346

[20] Bhowmick, S., Thakur, P. (2017). A Survey on Security Issues and Solutions in NoSQL Databases. International Journal of Computer Applications, 171(11), 32-37. doi:10.5120/ijca2017915327

[21] Tripathi, A., Shrivastava, P. (2017). Security Challenges in NoSQL Databases and their Solutions. International Journal of Computer Applications, 165(9), 1-5. doi:10.5120/ijca2017915289

[22] Sathiyaraj, A., Ramesh, R. (2017). A Review on Security Issues and Solutions in NoSQL Databases. International Journal of Computer Applications, 167(5), 1-4. doi:10.5120/ijca2017915590

[23] Shah, S. R., Ransing, V. (2018). Securing NoSQL Databases - A Systematic Review. International Journal of Computer Applications, 179(9), 7-10. doi:10.5120/ijca2018917626

[24] Hass, B., Maher, B. (2018). MongoDB in Action (2nd ed.). Manning Publications.

[25] Malavolta, I., Sorace, D., Spoto, F. (2017). Security Issues in NoSQL Databases: Analysis and Prevention. Procedia Computer Science, 112, 1839-1846. doi:10.1016/j.procs.2017.08.111

[26] Nayak, S., Manjula, D. (2015). A Study on Security Issues and Countermeasures in NoSQL Databases. International Journal of Scientific and Research Publications, 5(10), 1-5.

[27] Atreya, P., Kumar, S. (2018). A Review on Security Issues in NoSQL Databases. International Journal of Computer Applications, 180(4), 23-26. doi:10.5120/ijca2018917417

[28] Shetty, P. K., Singh, G. K. (2019). Security Challenges and Countermeasures in NoSQL Databases: A Review. International Journal of Advanced Research in Computer and Communication Engineering, 8(4), 358-363. doi:10.17148/ijarcce.2019.84113

[29] Kumar, V., Dhawan, S. (2018). An Insight into Security Issues in NoSQL Databases. Journal of Advanced Research in Dynamical and Control Systems, 10(3), 1084-1089.

[30] Bell, D., Lee, H., Dinh, H. T. (2017). Security Enhancements and Performance Evaluation of MongoDB. International Journal of Advanced Computer Science and Applications, 8(5), 254-261. doi:10.14569/ijacsa.2017.080537