MongoDB Injection Query Classification Model Using MongoDB Log Files As Training Data

Shaunak S. Perni[a*], Minal Naresh Shiroadkar[b]

[a] *Goa Business School, Goa University, Taleigao, Durgavado, Goa, 403206, India*
[b] *Goa Business School, Goa University, Taleigao, Durgavado, Goa, 403206, India*

Abstract

With previous studies creating models to classify MongoDB queries as benign or injection attack queries using only the query statement as input data. This study attempts the same problem of classifying queries using a model trained on query statements as well as additional variables found in the log files of an attacked MongoDB server

Since we did not find named log file data for training, an attack was simulated by randomly sending queries from a dataset containing an equal share of injection and benign queries to an empty MongoDB database. We extracted the log file, extracted all its features, and converted it into tabular form, thus creating an artificial dataset for the classification of injection queries on MongoDB log files.

The same dataset was then processed, cleaned, and explored, where we removed constant variables. We then tested the remaining variables for statistical significance in discriminating between injection and benign queries. Hence, we created a dataset of significant variables and trained machine learning models using an AutoML library, "FLAML", as well as 6 manually programmed models , which were then cross validated and evaluated. The study found that the best model was the "XGBoost limited depth" model with an accuracy of 71% that was produced by "FLAML"

All datasets and python notebooks are saved on the following git repository:
https://github.com/ShaunakPerniUniGoa/NoSQLInjectionDetection

---

* Corresponding Author
Email Address: msci.2105@unigoa.ac.in (Shaunak Perni)
Phone No: (+91) 8625946258

Generative AI content disclaimer

This paper has utilized generative AI tools, including ChatGPT-4, Grammarly, and Codium, for specific purposes such as code generation, debugging, and linguistic enhancements (e.g., grammar and style corrections). However, the authors affirm that the research, findings, and core content of this paper are solely their own work.

1 Background

The study of injection queries in NoSQL databases began in 2013 with the introduction of DIGLOSSIA, a tool capable of detecting both SQL and MongoDB NoSQL injection queries. Despite its innovation, DIGLOSSIA was limited by its rigid code format expectations, rejecting any user-based queries that deviated from this format. This constraint rendered it less adaptable across diverse business models (Son et al., 2013).

In 2015, a novel automata-based analysis approach was proposed to detect injection queries directed at MongoDB. While theoretically achieving a 100% detection rate, the methodology was confined to specific types of attacks, such as time-based and blind boolean attacks, thus offering limited security enhancements (Joseph and Jevitha, 2015)

By 2016, research had expanded to a comparative analysis of NoSQL and SQL databases against web-based vulnerabilities. The study, focusing on MongoDB and NoSQL, concluded that no effective solution had yet been developed to mitigate attacks on MongoDB, largely due to its widespread deployment in complex projects involving various tools and libraries (Abdalla et al., 2016)

In 2017 Methods were devised to measure the level of security of NoSQL databases from such attacks rather than detect injection queries on specific NoSQL databases or attack types, however the study concluded that the theoretical performance of this method is not applicable to protect against unique attacks that will be developed in the future (Algarni et al., 2017). In the same year a new parse tree based method called DND was proposed it was able to judge if an attack was an injection query or not and was able to store new attacks if it successfully detected them, the paper said it had achieved a test performance of 100% detection, however the training dataset was generated by the researchers hence the application is limited only to attacks discovered by the study (Ma et al., 2021).

In 2018 A study presented basic methods to identify vulnerabilities in NoSQL database to injection attacks for developers for (at the time's) current injection attacks, as the method was only developed for current day injection attack methods it was not guaranteed to be applicable to future attack methods (Sachdeva and Gupta, 2018). Another study presented a review of security and performance of MongoDB and other NoSQL database systems, it provide various insights in security and performance however for injection attacks it only recommend input cleaning as the only way to defend against injection attacks (Saxena and Sachdeva, 2018).

In 2019 the first ML/AI neural network based model was developed for MongoDB and CouchDB injection query detection, the model surpassed the current product Sqreen which had a detection rate of

36.25% compared to this model's 91.87% detection rate on MongoDB and 88.67% detection rate on CouchDB with a maximum error of 8% and 11% respectively with an F1 of 0.88, the input data was just the injection query statement (Ul Islam et al., 2019). Following this in 2021 a study proposed the use of AI based solutions specifically neural network based apporach to detect injection attacks in SQL and NoSQL based database systems, the study showed that their model had out performed most black list based models made in the previous decade (Alizadehsani, 2021). In the same year a new study came out with a model that had a maximum detection rate of 97.6% on any NoSQL database system using various low resource ML models (Mejia-Cabrera et al., 2021).

In 2022 a study conducted a study review of 3 database systems MongoDB, Redis and Cassandra (popular database systems at the time), it had concluded that MongoDB although the most popular of the 3 was the most vulnerable system due to it's simplicity (which also attributed to it's popularity), it also provided insights into how attacks can be conducted on each database system provided more insights in developing security solutions (Sanchez et al., 2021). In the same MURLi a tool was developed which stop malicious URLs to be sent to various NoSQL databases, stopping injections queries to even reach the database server before they could attack over the web, however direct queries were still able to reach the database server with an accuracy rate of 99.01% for NoSQL URLs, but it used deep learning models which were slow and costly to use. Alternatively another method was devised to stop occurrence of NoSQL injection attacks using encryption method specifically RSA and key pair values, however the study also mentions that this method has the aptitude and can be implemented on database systems, however no NoSQL database system including MongoDB has implemented such methods (Imam et al., 2022). Another study presented the SECURE-D a framework for web applications to detect and stop injection attacks on database servers and was able to detect in both SQL and NoSQL databases (Jithin and Subramanian, 2022). In the same year a new study had constructed a model with a higher performace of 0.92 F1 score improving over the previous studies in the field of low cost machine learning models (Praveen et al., 2022).

Finally, in 2024, an open dataset of 400 NoSQL queries for MongoDB was released, comprising 221 malicious and 179 benign queries. This dataset provided a valuable resource for future researchers aiming to develop AI and ML-based solutions for detecting injection queries in MongoDB (D· l· et al., 2024)

2 Introduction

This study attempts to advance the detection of MongoDB injection queries by employing deep learning models trained on a newly available open dataset. By integrating discovered system-based metrics with text-based features, the research aims to improve the accuracy and robustness of injection query classification. These advancements seek to enhance the automation of data collection and classification, contributing significantly to NoSQL, specifically MongoDB's database security.

3 Data Collection

We created a MongoDB 6.0.15 server on a Fedora 39 Linux Machine Kernel ver. 6.5 with an AMD Ryzen 7 57000U Processor with 8.0 GiB Memory on 1TB storage, using Mongosh 2.25 to send instructions to the database. We then set the profiling level on the server to 2 using the following command

```
db.setProfilingLevel(2,0.1)
```

Using the dataset, referred to as the “Query Dataset”, in this paper, we sent each query from the dataset to the database in this paper. Then, after executing all queries, we extracted the log file. We preserved log lines corresponding to the queries sent and removed the remaining variables from the file. We converted the remaining data to a tabular form, which will be called “log data” in this paper.

Each nested variable was expanded into individual columns. We joined the target attribute “label” from the Query Dataset to the log data using the column “Text” from the Query Dataset and the column “filter” from the log data.

Based on the filter column we engineered the variables based on

- Category of operator present in the filter as per MongoDB
- Type of selector present in the filter as per MongoDB
- Presence of a null operand
- Length of the query
- The query with only the MongoDB keywords present and the database variable names removed and the length of the same

After adding these engineered variables all constant variables were removed
The final structure of the collected data is shown in the following table (Table 1).

Field	Display Name	Data Type	Description
t	Timestamp	Timestamp	Timestamp of the log message in ISO-8601 format.
planSummary	Plan Summary	String	Plan used to execute the query.
planningTimeMicros	Planning Time in Microseconds	Float	Time taken to develop a query plan in microseconds.
cursorExhausted	Cursor Exhausted	Boolean	Whether the cursor was exhausted after execution.
queryFramework	Query Framework	String	Framework used to execute the query.
reslen	Response Length	Integer	Length of the query response in bytes.
cpuNanos	CPU Nanoseconds	Integer	CPU processing time in nanoseconds.

filter	Filter	String	Filter used in the query.
$eq	$EQ	Boolean	Whether the $eq operator is present in the query filter.
$gt	$GT	Boolean	Whether the $gt operator is present in the query filter.
$in	$IN	Boolean	Whether the $in operator is present in the query filter.
$ne	$NE	Boolean	Whether the $ne operator is present in the query filter.
$nin	$NIN	Boolean	Whether the $nin operator is present in the query filter.
$type	$TYPE	Boolean	Whether the $type operator is present in the query filter.
$mod	$MOD	Boolean	Whether the $mod operator is present in the query filter.
$regex	$REGEX	Boolean	Whether the $regex operator is present in the query filter.
$where	$WHERE	Boolean	Whether the $where operator is present in the query filter.
$elemMatch	$ELEM_MATCH	Boolean	Whether the $elemMatch operator is present in the query filter.
$size	$SIZE	Boolean	Whether the $size operator is present in the query filter.
$	Positional Operator	Boolean	Whether the positional $ operator is used.
>=	Greater Than or Equal To	Boolean	Whether the query uses a >= comparison.
<=	Less Than or Equal To	Boolean	Whether the query uses a <= comparison.
<	Less Than	Boolean	Whether the query uses a < comparison.
>	Greater Than	Boolean	Whether the query uses a > comparison.
selector_comparision	Selector Comparison	Boolean	Whether comparison selectors are used in the query.
selector_logical	Selector Logical	Boolean	Whether logical selectors are used in the query.

selector_element	Selector Element	Boolean	Whether element selectors are used in the query.
selector_evalutaion	Selector Evaluation	Boolean	Whether evaluation selectors are used in the query.
selector_array	Selector Array	Boolean	Whether array selectors are used in the query.
selector_bitwise	Selector Bitwise	Boolean	Whether bitwise selectors are used in the query.
projection	Projection	Boolean	Whether projection selectors are used in the query.
misc	Miscellaneous	Boolean	Whether misc selectors are used in the query.
selector	Selector	Boolean	Whether any selector operators are used in the query.
standard_logical	Standard Logical	Boolean	Whether standard logical operators are used in the query.
all_operators	All Operators	Boolean	Whether all operators are covered in the query analysis.
null_operand	Null Operand	Boolean	Whether null operands are used in the query.
regex_null_operand	Regex Null Operand	Boolean	Whether regex-based null operands are used.
text	Text	String	The filter query as it is
query_length_raw	Query Length (Raw)	Integer	Length of the query string as above.
keywords_only	Keywords Only	String	The query string but with only MongoDB query keywords
query_length_keywords_only	Query Length (Keywords Only)	Integer	Length of the query after extracting keywords only.
label	Label	Boolean	Label assigned to the query for identification or categorization.

Table 1: Final structure of the collected data

Please note due to technical issues some constant columns remained namely "cursorExhausted", "queryFramework" and "reslen"

4 Data Exploration

After collecting and pre processing the data (and the removal of the “cursorExhausted”, “queryFramework” and “reslen” manually) we separated the data into 2 types of variables binary, numerial and text, where the numerical variables were the variables with data type float or interger and binary variables were the variables with the data type boolean and the other string columns as text variables. The timestamp variable was removed from the training dataset because it represented artificially generated log data with no discernible pattern. Since it did not contribute meaningful or predictive information, including it in the model would have added noise rather than value. We only wanted to focus on the numerical and boolean variables hence removed the text variables.

4.1 Significance Testing

On the numerical variables we visualized the histograms and KDE densities

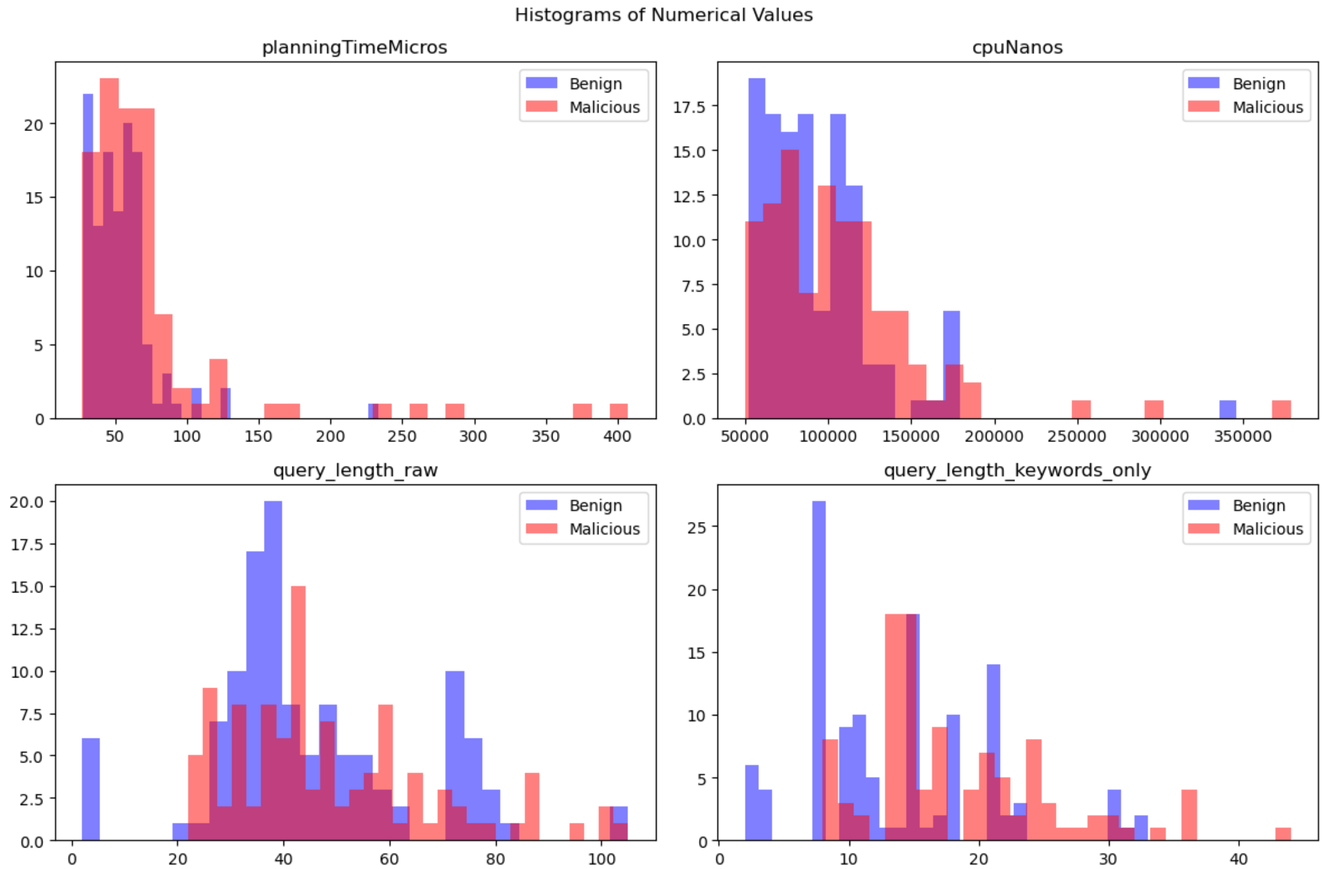

Figure 1: Histograms of numerical variables, blue represents Benign queries sample and red represents Injection queries sample, clockwise from Top left corner, planningTimeMicros, cpuNanos, query_length_raw, query_length_keywords_only

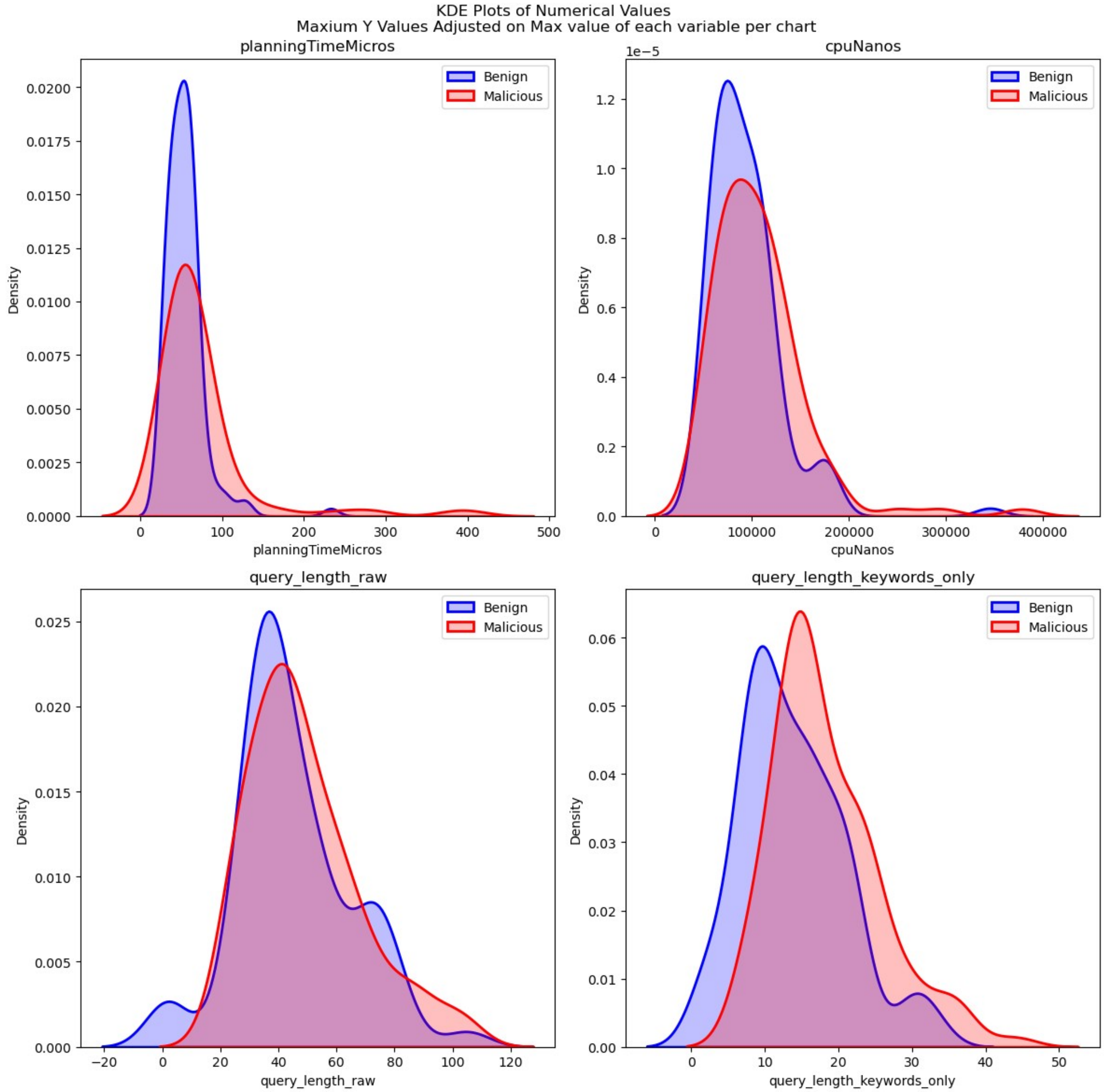

Figure 2: Kernel density estimation plots of numerical variables, blue represents Benign queries sample and red represents Injection queries sample, clockwise from Top left corner, planningTimeMicros, cpuNanos, query_length_raw, query_length_keywords_only

And then conducted a Mann-Whiteney U test of significance from the initial numerical columns [planningTimeMicros, cpuNanos, query_length_raw, query_length_keywords_only] only planningTimeMicros and query_length_keywords_only were found to be significant at 0.01 signifance hence these 2 variables were kept and the rest were removed

On the boolean variables we conducted a ChiSq test to determine the significance of the boolean variables from the initial boolean columns ['$eq', '$gt', '$in', '$ne', '$nin', '$type', '$mod', '$regex', '$where', '$elemMatch', '$size', '$', '>=', '<=', '<', '>', 'selector_comparision', 'selector_logical', 'selector_element', 'selector_evalutaion', 'selector_array', 'selector_bitwise', 'projection', 'misc', 'selector', 'standard_logical', 'all_operators', 'null_operand', 'regex_null_operand'] only the variables $ne, '$' were found to be significant at 0.01 significance hence these 2 variables were kept and the rest were removed

4.2 Separability Analysis

After determining the significant variables the columns were combined into 1 table which will now be refered to as the "final dataset" included with the "label" variable. After which we conducted separability analysis to determine if the groups were separable enough via visualization

4.2.1 LDA

The following figure shows the LDA visualization

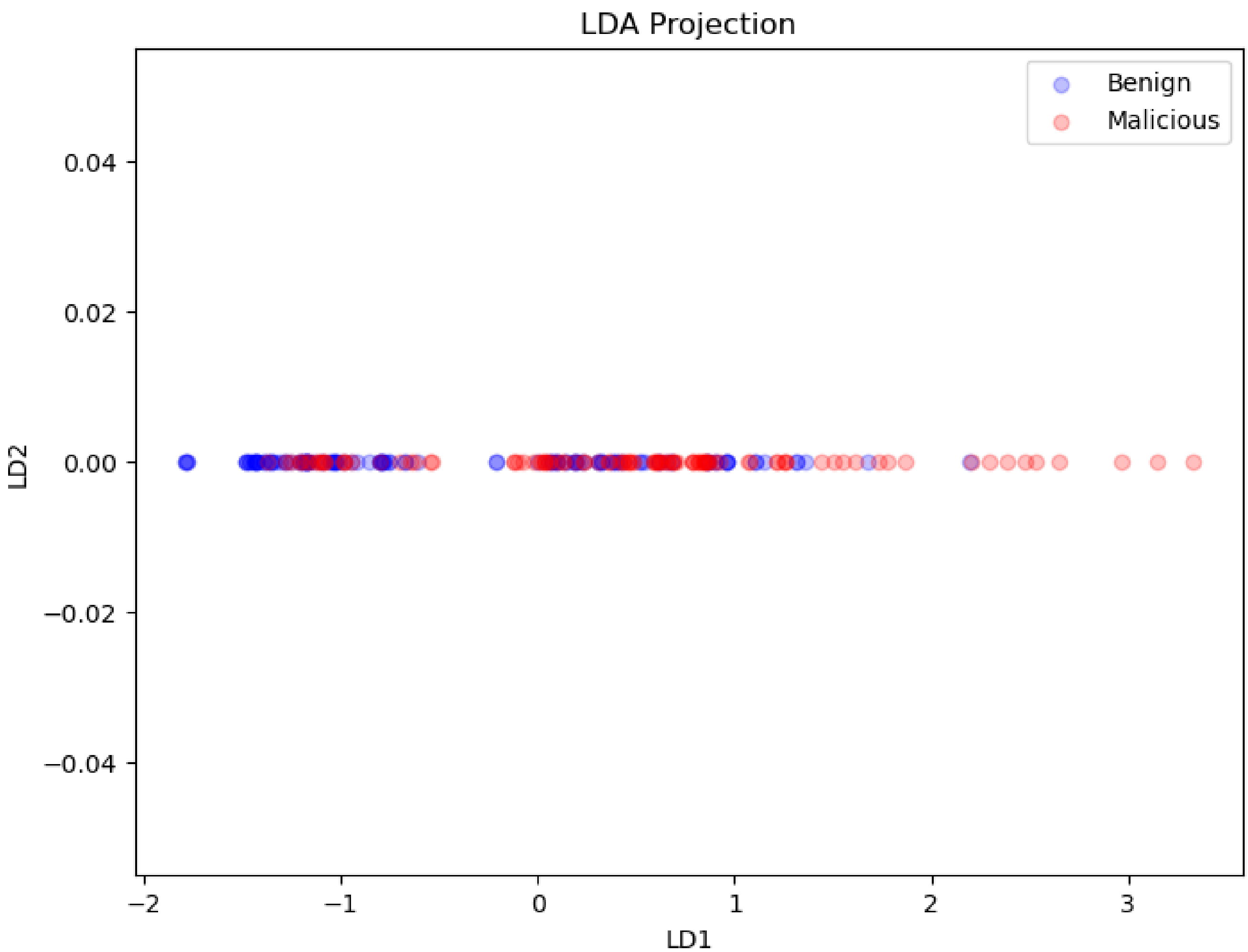

Figure 3: LDA Projection Graph , Blue circles represent benign query data points and Red circles represent injection query data points

4.2.2 PCA

The following figure shows the PCA visualization

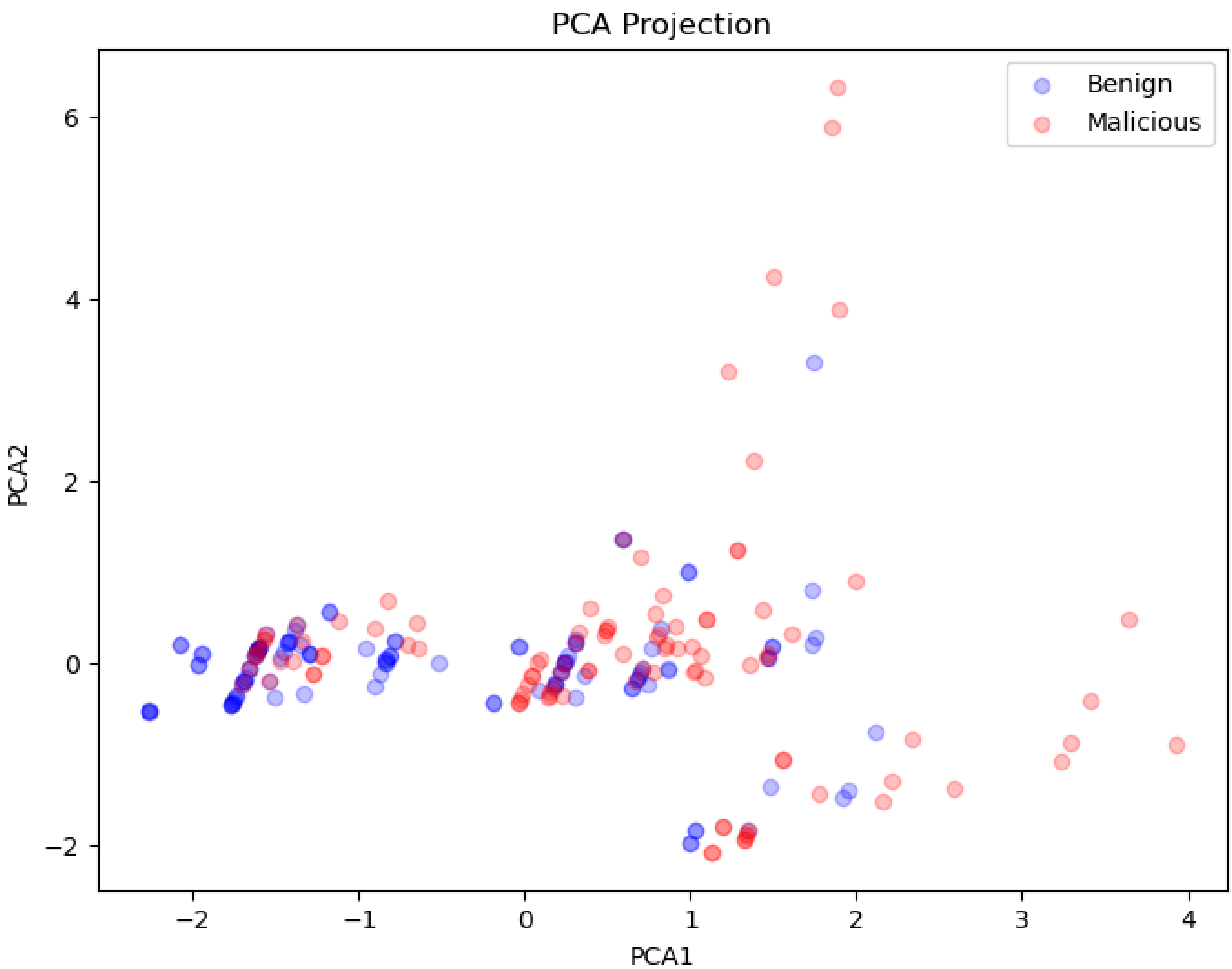

Figure 4: PCA Projection Graph , Blue circles represent benign query data points and Red circles represent injection query data points

4.2.3 t-SNE

The following figure shows the t-SNE visualization

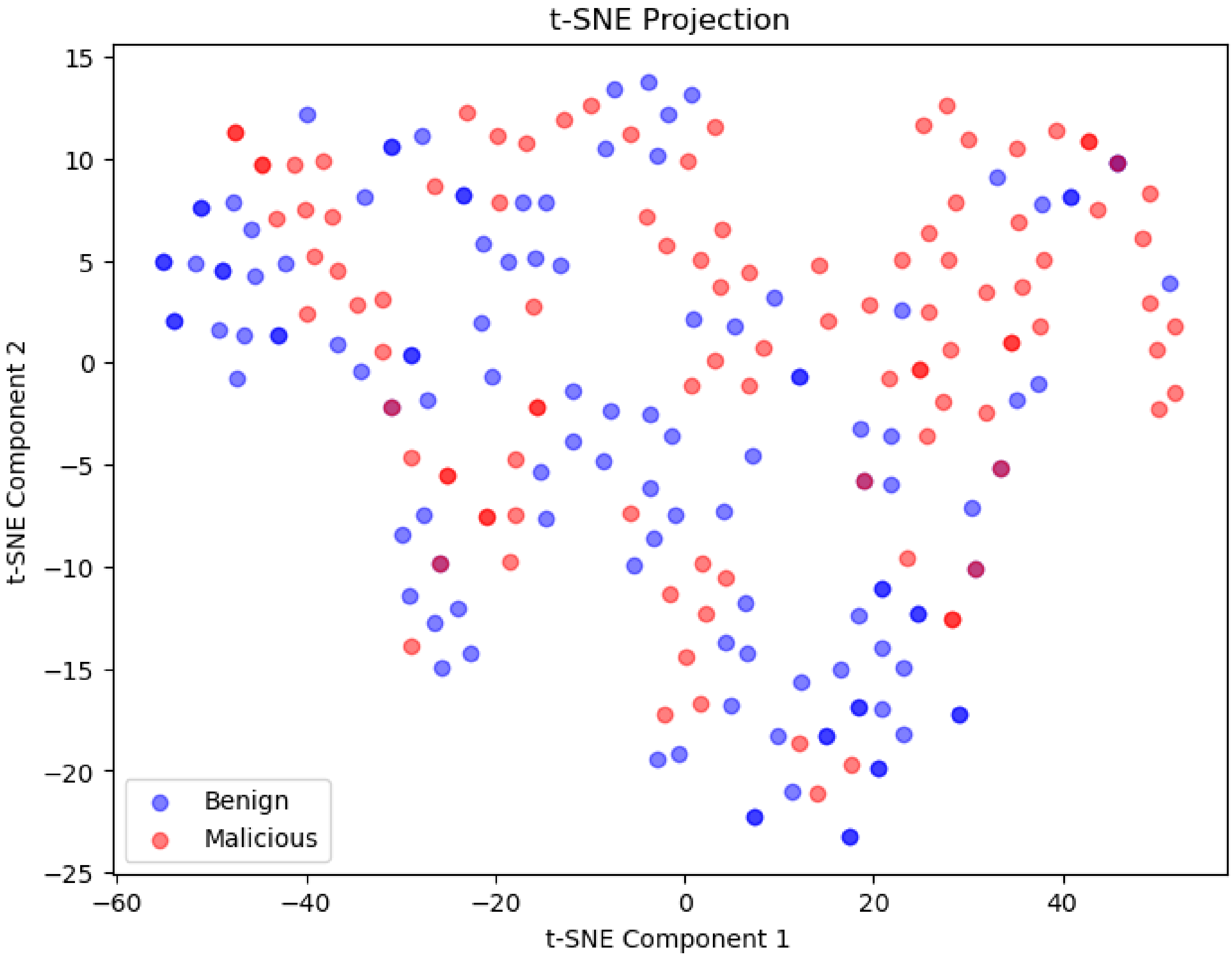

Figure 5: t-SNE Projection Graph , Blue circles represent benign query data points and Red circles represent injection query data points

We then conducted a K-means test to get the clusters of each sample , the number of clusters are manually determined by trial and error we ended up with 13 clusters for samples with label = 0 and 15 samples with label = 1,we then created polygons of each cluster and overlapped the clusters to determine the total amount of overlapped.

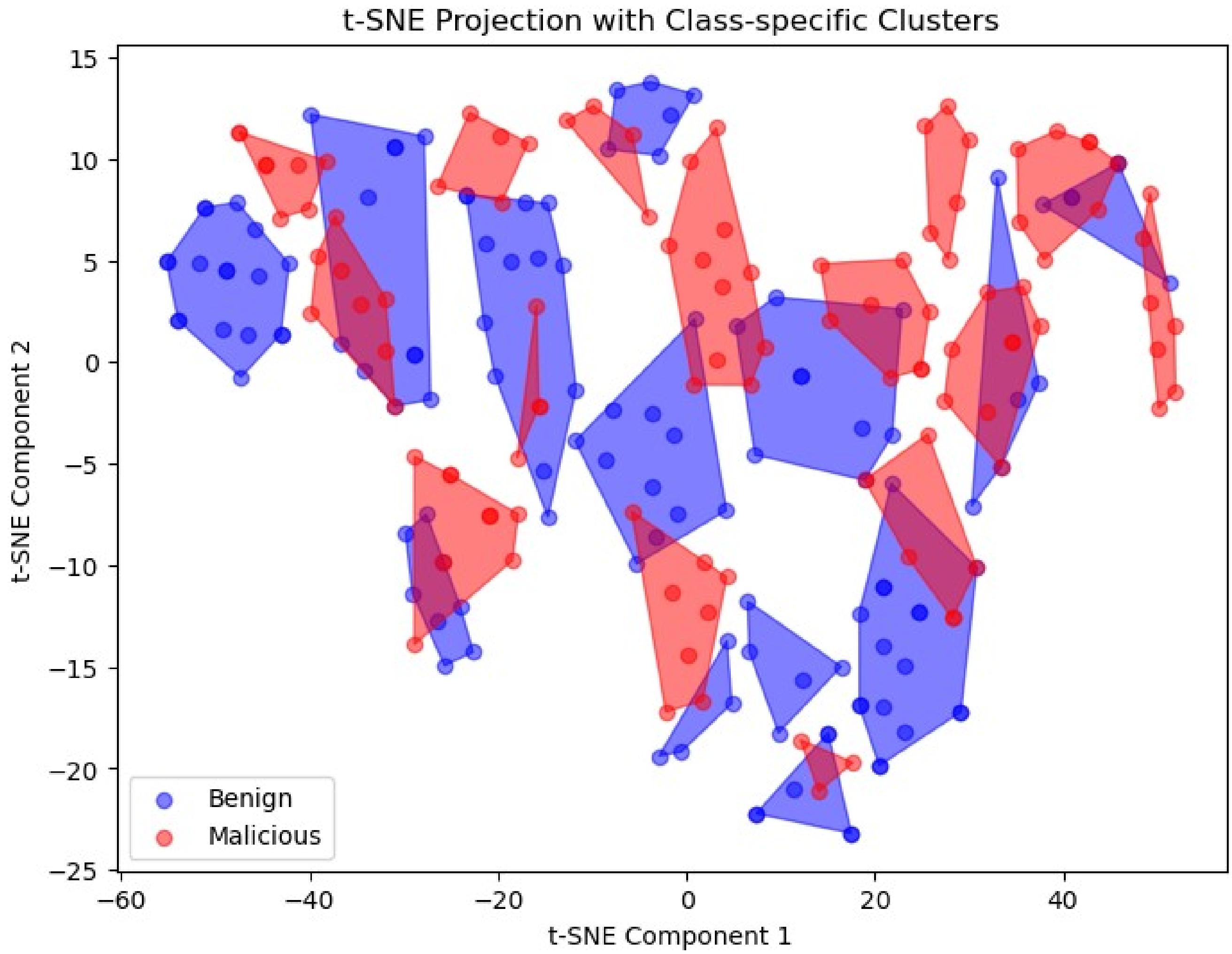

Figure 6: t-SNE Projection Graph with polygons based on K means clustering , Blue circles and polygons represent benign query data points and region and Red circles and polygons represent injection query data points and regions

We believe this may give an idea of how much maximum accuracy we should expect from any trained model, from the area calculation there was a 13.54% of overlapping between thus giving us an estimated 86.46% maximum accuracy achievable with the dataset

The final dataset’s structure is as follows

Field	Display Name	Data Type	Description
$ne	$NE	boolean	If a $ne operator exists in the query filter
planningTimeMicros	Planning Time Microseconds	float	Time taken to develop a query plan in microseconds.
$	$	boolean	If the $ character exists in the query filter
query_length_keywords_only	Query Length with only Keywords	int	The length of the query filter string if only variables names are removed
label	Label	boolean	If the query is a injection query or not

Table 2: Structure of data only with statistically significant variables

5 Model Formulation

The general formula for the model is as follows

Lable ~ $ne + planningTimeMicros + $ + query_length_keywords_only

The model would be a classification model, i.e. to classify if given query properties such as the above if the query is an injection query or a benign query

We decided to test 6 model algorithms and FLAML's chosen model

The algorithms chosen were:

1. Logistic Regression
2. Random Forest
3. Support Vector Machine
4. K-Nearest Neighbors
5. Decision Tree
6. Naive Bayes

FLAML AutoML model was configured to classification with a time budget of 60 seconds with 5 dataset splits for cross validation

5.1 Model Training and Testing

The final dataset was divided into 3 parts of Training, Testing and Validation which respectively were 60%, 20%, 20%. Model 1-6 except the FLAML AutoML model were trained on these sets with a random seed going from 1 – 51 using cross validation , for each iteration the following results were recorded.

- The name of the Model in column “Classifier”,
- The Accuracy,
- Precision,
- Recall
- and F1 Score

and each metric was cross validated with 5 dataset splits and recorded. After all iterations the average Accuracy, Precision, Recall and F1 Score from each iteration was calculated

For the FLAML AutoML model, it was given the training dataset and set to 5 dataset splits

6 Evaluation

After training the following results were recorded

Model	Accuracy (in %)	Precision (in %)	Recall (in %)	F1 Score (in %)
Logistic Regression	65.45% (avg)	67.73% (avg)	66.73% (avg)	66.64% (avg)
Random Forrest	67.23% (avg)	68.38% (avg)	70.36% (avg)	67.97% (avg)
Support Vector Machines	55.97% (avg)	59.55% (avg)	46.36% (avg)	50.00% (avg)
K-Nearest Neighbors	53.25% (avg)	53.59% (avg)	63.27% (avg)	56.66% (avg)
Decision Tree	66.32% (avg)	67.49% (avg)	66.91% (avg)	66.71% (avg)
Navie Bayes	59.87% (avg)	72.00% (avg)	39.27% (avg)	49.28% (avg)
FLAML Best Estimator (XBG Limited Depth)	73.00%	75%	73%	74%

Table 3: Cross Validated and average evaluation metrics of models (1-5) and cross validated and best evaluation metrics of FLAML AutoML model against tested data of different randomization samples

Hence we find that the XBG Limited Depth algorithm has performed the best with the highest accuracy, precision, recall and F1 score among all other models.

7 Conclusion

The outputs of this project demonstrates that it is possible to make a model to classify injection queries sent to a MongoDB server based on the log data, even if the event is after an attack on the server for the log entry to be created. Hence such models may assist in creation of dataset by identify log lines which may contain a injection query to train models to stop injection queries before they execute or maybe models that stop execution of the query mid process such as the amount of time taken for creating a plan on MongoDB as demonstrate as a significant variable in this dataset of this paper.

In addition we also determined some variables form the logs to be statistically significant to discriminate between injection and benign queries.

However due to the dataset being artificially generated, we believe following the same methodology of this paper from EDA to model formulation and to the results of the evaluation of models. The researcher may find different results including, different significant variables to discriminate between injection and benign queries, a different significant engineered variables or even different separability between injection and benign query features

8 Limitations

The data collected and used for this paper is artificial and may not reflect data found in the real world

The removal of certain variables due to them being constant or too randomized such as the timestamp variable may play as a significant variable if used on a real world data

The paper is only limited to statistically significant variables being used for the model. For a deep learning model all variables may be required

9 References

Abdalla, H.B., Li, G., Lin, J., Alazeez, M.A., 2016. NoSQL injection: Data security on web vulnerability. International Journal of Security and its Applications. https://doi.org/10.14257/ijsia.2016.10.9.07

Algarni, A., Alsolami, F., Eassa, F., Alsubhi, K., Jambi, K., Khemakhem, M., 2017. An open tool architecture for security testing of NoSQL-based applications. Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA. https://doi.org/10.1109/AICCSA.2017.44

Alizadehsani, Z., 2021. Proposing to use artificial neural networks for NoSQL attack detection. Advances in Intelligent Systems and Computing. https://doi.org/10.1007/978-3-030-53829-3_29

D· l·, R., H· v·, A., Honnavalli, P.B., S·, N., 2024. The MongoDB injection dataset: A comprehensive collection of MongoDB - NoSQL injection attempts and vulnerabilities. Data in Brief 54, 110289. https://doi.org/10.1016/j.dib.2024.110289

Imam, A.A., Basri, S., Gonzalez-Aparicio, M.T., Balogun, A.O., Kumar, G., 2022. NoInjection: Preventing Unsafe Queries on NoSQL-Document-model Databases. Proceedings of 2022 2nd International Conference on Computing and Information Technology, ICCIT 2022. https://doi.org/10.1109/ICCIT52419.2022.9711654

Jithin, V.O., Subramanian, N., 2022. SECURE-D:Framework For Detecting and Preventing Attacks in SQL and NoSQL Databases. 10th International Symposium on Digital Forensics and Security, ISDFS 2022. https://doi.org/10.1109/ISDFS55398.2022.9800805

Joseph, S., Jevitha, K.P., 2015. An Automata Based Approach for the Prevention of NoSQL Injections, in: Abawajy, J.H., Mukherjea, S., Thampi, S.M., Ruiz-Martínez, A. (Eds.), Security in Computing and Communications. Springer International Publishing, Cham, pp. 538–546. https://doi.org/10.1007/978-3-319-22915-7_49

Ma, Z., Ma, H., Gao, X., Cai, T., Zhang, X., 2021. BLSTM-based source code vulnerability detection visualization system. Proceedings of SPIE - The International Society for Optical Engineering. https://doi.org/10.1117/12.2624204

Mejia-Cabrera, H.I., Paico-Chileno, D., Valdera-Contreras, J.H., Tuesta-Monteza, V.A., Forero, M.G., 2021. Automatic Detection of Injection Attacks by Machine Learning in NoSQL Databases, in: Roman-Rangel, E., Kuri-Morales, Á.F., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Olvera-López, J.A. (Eds.), Pattern Recognition. Springer International Publishing, Cham, pp. 23–32. https://doi.org/10.1007/978-3-030-77004-4_3

Praveen, S., Dcouth, A., Mahesh, A.S., 2022. NoSQL Injection Detection Using Supervised Text Classification, in: 2022 2nd International Conference on Intelligent Technologies (CONIT). Presented at the 2022 2nd International Conference on Intelligent Technologies (CONIT), pp. 1–5. https://doi.org/10.1109/CONIT55038.2022.9848017

Sachdeva, V., Gupta, S., 2018. Basic NOSQL Injection Analysis And Detection On MongoDB, in: 2018 International Conference on Advanced Computation and Telecommunication (ICACAT). Presented at the 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), pp. 1–5. https://doi.org/10.1109/ICACAT.2018.8933707

Sanchez, R.A.G., Bernal, D.J.M., Parada, H.D.J., 2021. Security assessment of Nosql Mongodb, Redis and Cassandra database managers. 2021 7th Congreso Internacional de Innovacion y Tendencias en

Ingenieria, CONIITI 2021 - Conference Proceedings.
https://doi.org/10.1109/CONIITI53815.2021.9619597
Saxena, U., Sachdeva, S., 2018. An insightful view on security and performance of NoSQL databases. Communications in Computer and Information Science. https://doi.org/10.1007/978-981-10-8527-7_54
Son, S., McKinley, K.S., Shmatikov, V., 2013. Diglossia: Detecting code injection attacks with precision and efficiency. Proceedings of the ACM Conference on Computer and Communications Security. https://doi.org/10.1145/2508859.2516696
Ul Islam, M.R., Islam, Md.S., Ahmed, Z., Iqbal, A., Shahriyar, R., 2019. Automatic Detection of NoSQL Injection Using Supervised Learning, in: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). Presented at the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), pp. 760–769.
https://doi.org/10.1109/COMPSAC.2019.00113