Behavior-Based Anomaly Detection for Malicious SQL Queries in Privileged Environments

Ramez Hatab
ramezhatab@utexas.edu
UT Austin, USA

Abstract

Detecting malicious SQL statements within elevated-privilege environments poses a clear challenge to database security. Insider threats and privilege escalation techniques enable attackers to bypass traditional application-layer defenses and execute harmful operations directly on the database. This leaves critical vulnerabilities that allow adversaries to perform reconnaissance, exfiltration, and alteration of data completely undetected.

In this paper, I analyze the challenge of detecting malicious SQL queries executed in elevated privilege environments. Separating my project from previous work that focuses on labeled datasets and predefined rules, I present **Deadeye**. Deadeye is a behavior-based anomaly detection model that identifies malicious queries at execution time by analyzing the expected behaviors and structures of SQL statements. The tool employs query normalization and feature extraction techniques to create a baseline of legitimate SQL traffic and uses an Isolation Forest model to detect anomalies.

I demonstrate the effectiveness of Deadeye through simulating and collecting traffic from an industry-level application, followed by embedding malicious queries to assess the model's performance. The tool's ability to detect anomalies within real data is evaluated, and results show that Deadeye effectively detects malicious queries with high accuracy and a low false-positive rate. This work highlights the importance of behavior-based analysis for mitigating insider threats and presents a scalable solution for anomaly detection in modern database systems.

ACM Reference Format:
Ramez Hatab. 2024. Behavior-Based Anomaly Detection for Malicious SQL Queries in Privileged Environments. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Malicious SQL queries pose a significant threat to database security, specifically in systems where adversaries gain elevated privileges. These statements can perform reconnaissance, exfiltration, or manipulation on sensitive data and go completely undetected in certain environments. Insider threats and privilege escalation attacks are particularly worrisome, as they enable attackers to interact directly with the database, circumventing safeguards like input sanitization and parameterized queries.

While SQL injection (SQLi) remains one of the most well-known and dangerous attack vectors, leading to data exposure, theft, or deletion, the broader challenge lies in detecting malicious queries that occur after initial compromise. For instance, consider the following query execution from an adversary with root privileges:

```
DROP TABLE users; --
```

This query, when executed, deletes the entire `users` table and omits whatever SQL code was after it. This single statement illustrates the destructive potential of these threats and is just one of many examples of possible attacks that can emerge from insider threats, compromised credentials, or privilege escalation techniques. Such actions are difficult to distinguish from legitimate administrative queries, making real-time detection of anomalous behavior a critical factor in database security.

Despite advances in database security, existing solutions often overlook the detection of malicious queries in privileged environments. Many rely on supervised learning, which requires balanced and carefully labeled datasets of both benign and malicious queries. However, these datasets are often incomplete, overlooking a wide variety of possibilities such as obfuscated queries, or those that mimic legitimate traffic in structure. Static rule-based approaches are similarly inadequate as they cannot adapt to novel or obfuscated threats. This gap underscores the need for a behavior-based approach capable of identifying deviations from normal database activity, independent of predefined rules or labeled training data.

In an effort to address this oversight, I present Deadeye, a behavioral analysis framework that inspects SQL queries at execution time. The framework leverages a novel approach

that combines query normalization, structural pattern analysis, and unsupervised machine learning to detect anomalous behavior. The key insight behind this approach is the observation that legitimate SQL traffic typically follows predictable patterns. Queries often differ only in values or literals while maintaining consistent structures and operations. By normalizing queries and extracting structural and behavioral features, Deadeye creates a baseline for legitimate activity, enabling it to identify deviations that signify potential threats. This approach not only addresses insider threats, but also mitigates risks posed by adversaries with elevated privileges executing sophisticated SQL statements.

Deadeye uses a simple three-step method to process SQL queries in real time:

1. Normalize SQL queries to remove noise including literals and column or table names.
2. Extract structural and behavioral features from payloads.
3. Use an isolation forest model to detect any deviations from learned expected behavior.

To assess its effectiveness, Deadeye was trained on a dataset of benign GitLab SQL traffic for the purposes of this research and tested against a combined dataset that included generated malicious queries. The experimental results demonstrate the framework's ability to accurately distinguish between benign and anomalous SQL statements, achieving high recall with a low false-positive rate. Notably, Deadeye is designed solely for detecting malicious SQL queries under the assumption that adversaries have already gained root privileges. Its primary focus is on post-compromise activities, such as schema alterations and data exfiltration.

The remainder of this paper is organized as follows. Section 2 discusses the motivation for the project and why it is relevant. The system under test as well as detailed discussion on data collection and generation is discussed in Section 3, and the results of testing Deadeye are presented in Section 4. Section 5 discusses other research in the problem area, and finally Section 6 concludes the paper with a discussion of key findings.

2 Motivation

While a variety detection systems exist, very few take a machine learning approach and even less attempt this challenge using unsupervised learning. As a result, challenges persist in adapting defenses to evolving threats and detecting zero-day attacks.

This paper takes information and inspiration from Sherlock [4], a framework that is capable of quantifying and improving the effectiveness of hardware-based mobile malware detectors (HMDs). Kazdagli et al. highlights the necessity of having a form of behavior-based analysis system, utilizing a one-class SVM (ocSVM) to establish a baseline for normal mobile activity and detect any anomalies in various power

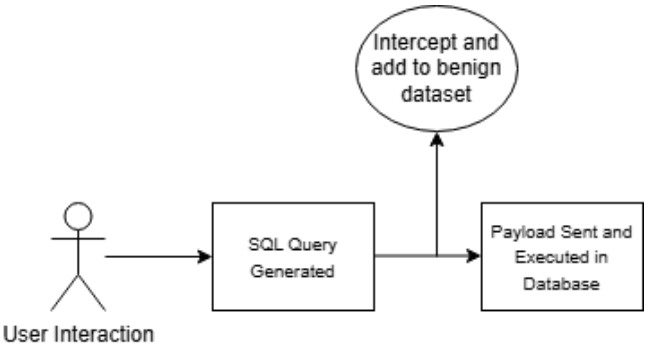

Figure 1. Flow of how benign data is simulated and collected to form training dataset.

trace patterns. The researchers use this methodology to improve existing unsupervised methods for detecting malware with a focus on payload-centric machine learning features.

Inspired by Sherlock, I decided to apply this same principle to SQL databases, creating my own unsupervised anomaly detection model as opposed to actively attempting to improve existing detectors. Extending this, my work emphasizes execution of malicious queries in elevated privilege environments, and utilizes an isolation forest over a ocSVM.

3 System Under Test

The system under test, Deadeye, is designed to detect anomalous SQL queries in real-time by inspecting their structural and behavioral features. This framework utilizes various feature extraction techniques, then an isolation forest model to accurately identify anomalies suggestive of insider threats or a possible compromised account. This remaining parts of this section will outline our environment, data collection, and system flow.

3.1 Data Collection

A defining feature of Deadeye is its ability to detect anomalies by training solely on legitimate traffic. For this research project, I opted to base my work in GitLab, setting up an instance in a Docker container.

3.1.1 Benign Data. To collect benign data, I created a Python script simulating user interactions in GitLab such as creating users, projects, merge requests, issues, comments, etc. All of these actions generated SQL queries which interacted with the PostgreSQL database, and I was able to capture each of them using verbose logging techniques. This general process can be visualized in Figure 1. The Python script was executed for five minutes and produced 181,000 SQL queries, forming my training dataset for my model.

3.1.2 Malicious Data. In order to evaluate Deadeye's ability to detect malicious queries as anomalies, I created a SQL

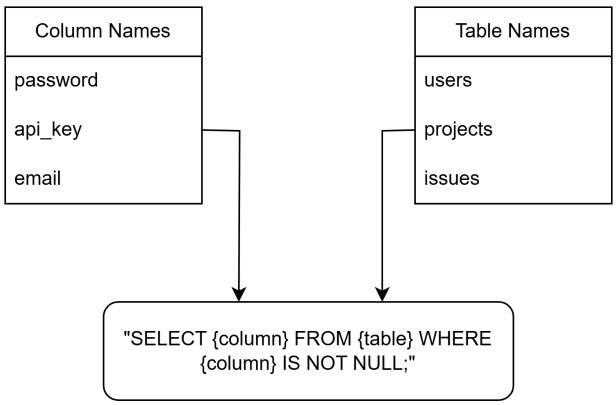

Figure 2. Malicious query skeleton with possible dynamic options for table and column names, determined at runtime.

statement generator that produces a random output after every execution. This automated the creation of attack vectors for testing my model, and allowed me to implement realistic anomalous payloads as well.

The script ensures variability by having a predefined set of payload "skeletons", and randomizing the internal values and literals with relevant GitLab ones such as existing table and column names. In Figure 2, this process is outline and a query that represents a realistic attack scenario of an adversary performing data reconnaissance on a table, identifying any non-null columns.

Alongside data reconnaissance, exfiltration and alteration, my generator includes brief SQL injection examples. It does this by simply taking some full statements from the benign dataset (legitimate traffic), and appending some common SQLi payload at the end such as `OR '1'='1'; --`. This not only further simulates real-world attacks, but stress tests my model to see even if a majority of the malicious statement resembles legitimate traffic, can it pick up on an appended injection payload.

3.2 Feature Engineering

Feature engineering plays a pivotal role in creating an accurate anomaly detection model. Traditional SQL queries have large amounts of noise which impact model performance, especially those using unsupervised learning. To address this issue, Lu et al. outlined an interesting principle of query normalization ad focusing on the structure and embeddings to implement SQL injection attack detection models [6]. Their approach demonstrated the effectiveness of focusing on query structure over content, laying a foundation for advanced detection techniques. Building on this work, I utilize query normalization amongst other feature selection techniques to emphasize malicious query detection in privileged environments.

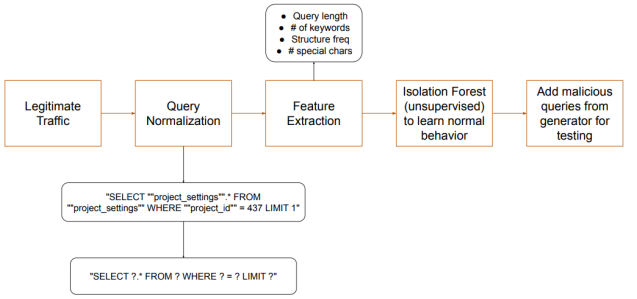

Figure 3. Deadeye system flow including query normalization and feature selection techniques.

3.2.1 Query Normalization.

Lu et al. discussed a relevant normalization technique to focus on the structure of queries. I applied this to my research by representing benign queries without their literals and identifiers. This process can be visualized in Figure 3.

This methodology abstracts away specific values and focuses on the query's structure, making it easier to detect deviations from expected payloads. Since GitLab queries will always maintain the same structure with different identifiers and literals, we can effectively use this attribute to train our isolation forest and effectively identify a majority of malicious queries.

3.2.2 Miscellaneous Feature Selection.

In addition to query normalization, Deadeye incorporates four additional features to capture the structural and behavioral attributes of SQL queries. These features further the model's ability to detect deviations indicative of malicious behavior:

1. **Query Length**: Gets the total number of characters in a query. Malicious queries often deviate in length compared to benign ones due to complexity or simplicity.
2. **Structure Frequency**: Captures how often a specific query structure appears in the training dataset, allowing us to detect deviations (anomalies).
3. **Number of Special Characters**: Counts occurrences of characters such as `'`, `"`, `;`, and `#`.
4. **Number of SQL Operations**: Counts the number of operations like `SELECT`, `INSERT`, `DELETE`, and `UPDATE` in a query.

These features were selected after experimentation with various others, as they produce the best results in maximizing our precision and recall of benign and malicious statements. This feature selection process is outlined in the diagram present in Figure 3, as well as its integration into the overall system flow of Deadeye.

3.3 Model Training

Unsupervised models are excellent for tasks where you have unlabeled data and want to detect anomalies that deviate from such a baseline [8]. This insight presented exactly the

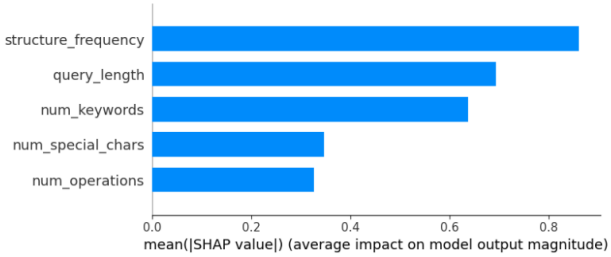

Figure 4. Graph showing mean SHAP value of each feature and its impact on model predictions and performance.

solution I was looking for, as well as a novel perspective on SQL anomaly detection. Unsupervised learning would allow detection of anomalies in a realistic environment like GitLab with just access to their legitimate logs. The need for a large red team or access to a highly extensive and detailed labeled dataset is eliminated, and zero-day threats become less of a concern.

The isolation forest was chosen for this research for those exact purposes. The algorithm models normal query behavior by learning the patterns and structures within the benign dataset, enabling it to flag outliers indicative of potential threats. Isolation forest specifically uses decision trees that partition on random values within feature sets, meaning inliers will be deep within the tree (longer path length) and outliers will be shallow (shorter path length) [1].

Our model was trained exclusively on the normalized and feature-engineered dataset derived from the simulated GitLab dataset (Section 3.1.1). A valuable insight into feature importance for our models performance can be seen in Figure 4, with structure frequency which was extracted using the normalization technique outlined (Section 3.2.1) leading the way. Query length and the number of keywords in a statement also proved to be extremely valuable. This graph was generated using SHAP values, which are determined by iteratively training the model and comparing its predictions with and without particular features. This is done iteratively for each feature and each sample in the dataset [7].

3.3.1 Model Optimization. Isolation forest performed well without any tuning needed, but utilizing hyperparameter optimization the best possible accuracy was achieved.

The most impactful hyperparameter was *Contamination*. Contamination is meant to reflect an estimated proportion of anomalies in real-world traffic. After performing a simple GridSearch with a three-fold cross validation, the optimal contamination parameter was set to 0.019.

The other hyperparameters including number of trees and maximum samples were deemed best at the default setting, so those remained unchanged. The testing could now be conducted with the optimal model.

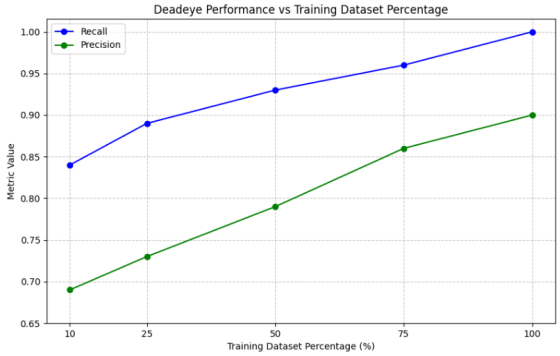

Figure 5. Graph showing both recall and precision of detecting malicious SQL queries with varying percentages of the training dataset to train Deadeye.

4 Experimental Results

The experimental evaluation of Deadeye was conducted to assess its performance in detecting malicious SQL queries while maintaining a low false-positive rate. The results highlight the effectiveness of the proposed framework in identifying deviations from legitimate database activity, including scenarios where malicious traffic mimics legitimate instances and SQLi payloads are appended.

The following tests were performed: evaluating Deadeye's performance with differing percentages of the training set available, as well as its performance with the full dataset and a realistic amount of malicious queries for testing.

4.1 Differing Training Set Percentages

Setting up a Python script, the optimal Deadeye model was tested with 10%, 25%, 50%, 75% and finally 100% of the training dataset selected at random. The recall and precision for malicious queries were of particular interest, and the results of this experiment can be seen in Figure 5. As expected with unsupervised learning, our model becomes more accurate with increasing amounts of training data. For this test, 50 malicious queries were injected into 500 benign queries to create our testing set, and the results show that after 100% training data was reached we achieved a recall of 100% and a precision of 90%. At it's peak, Deadeye only saw 5 false-positives.

With this training data being retrieved from GitLab, one can safely predict that from the results seen in the graph if around one hour of traffic was logged, Deadeye could achieve close to perfect results.

4.2 Realistic Testing

To simulate a possible real-world attack scenario, such as an insider threat, Deadeye was tested with a highly imbalanced dataset containing a large discrepancy in the ratio of benign to malicious queries. Specifically, the model was given 5000

benign queries and 5 malicious queries, representing a 1000:1 ratio.

However, the results revealed a limitation in the model's configuration: the contamination parameter, which defines the expected proportion of anomalies in the dataset, was not low enough to reflect the true anomaly rate in this scenario. As a result, Deadeye struggled to differentiate anomalies effectively, often misclassifying malicious queries as benign. This highlights a key challenge in applying unsupervised learning models like Isolation Forest to environments with extremely rare anomalies—an underestimation of the anomaly rate can significantly reduce recall for malicious activity.

The inability to adjust the contamination parameter to sufficiently low levels underscores a critical trade-off between model sensitivity and false-positive rates. Future iterations of this framework will explore adaptive contamination thresholds or alternative models such as an ocSVM as seen in Kazdagli et al. [4].

5 Related Work

5.1 AMNESIA

Halfond et al. utilize AMNESIA, a tool that detects and prevents SQL injection attacks by combining static analysis and runtime monitoring. The framework functions by first analyzing application code to determine all possible legitimate query templates that the application could generate. Then, it compares these retrieved templates against dynamically generated queries at runtime to detect malicious injections that deviate from the allowed patterns [3].

These researchers provide a very valuable lens through which Deadeye functions, which is learning *expected* query behavior. This was a large basis of the work done in this paper, and complements it by focusing specifically on the application layer with Deadeye existing further back in the flow and honing in on post-compromise possibilities.

5.2 Anomaly Detection Using Supervised Learning

Gupta et al. explores the application of various supervised learning models, including Naive Bayes, SVMs, and CNNs, to detect SQLi patterns [2].

The researchers display the effectiveness of these techniques in identifying SQL injection attacks, particularly through static and dynamic query features. However, the reliance on labeled datasets for training presents a key limitation, particularly in adapting to find zero-day threats, or if an entity is unable to access such a comprehensive dataset for training.

Unlike traditional supervised methods, Deadeye employs an unsupervised Isolation Forest model, allowing it to detect anomalies without relying on predefined labels for malicious queries. Additionally, Gupta et al. focuses on SQL injection threats, with Deadeye honing on specifically on malicious queries with some discussion of detect SQLi attacks.

5.3 User Behavior ocSVM Anomaly Detection

Li et al. explores database environments which is a critical insight for this paper, as we are working with relational database queries. More specifically, the researchers base an audit technology around a ocSVM to assess user behavior [5].

The authors utilize audit logs for their training dataset, extracting relevant features through syntax parsing and numerical processing methods. These audit logs reflect the rules of "normal" behavior for their long term operations, so they train a ocSVM to construct their detection system. they use this ocSVM model to realize the security audit of database user access behavior.

Achieving an average of 92% detection accuracy amongst three different kinds of abnormal behavior, Li et al. provide valuable insight for future work and applications of Deadeye. It becomes clear that implementing an unsupervised model and achieving high performance metrics with it becomes possible, especially in real-world scenarios such as audit logs.

6 Conclusions

In this paper, I presented Deadeye, a behavior-based anomaly detection framework designed to detect malicious queries. With a large focus on post-compromise scenarios including insider threats and privilege escalation, Deadeye addresses a critical gap that exists in database security. Unlike popular supervised learning approaches, Deadeye attempts to find the ability to detect zero-day threats using unsupervised learning to identify any deviations in query behavior or structure from benign traffic.

Through extensive testing, experimental results yielded high precision in classifying both benign and malicious statements as well as a near-perfect recall percentage for both classes. Furthermore, the incorporation of features such as query normalization, structure frequency, and behavioral analysis underscored the importance of leveraging query context rather than static content.

Despite its prevalent success under certain circumstances, Deadeye falls short in critical areas including handling scenarios with extremely low anomaly rates due to contamination parameter constraints. Future iterations of this framework will assess this issue and explore various models as well as far more complex payloads to ensure optimal functionality of Deadeye.

Overall, this work emphasizes the importance of behavior-based anomaly detection in securing modern database systems. Deadeye offers a scalable solution that is not only capable of detecting malicious queries, but also provides a foundation for future research in anomaly detection for privileged environments.

References

[1] W. Chua, A. L. D. Pajas, C. S. Castro, S. P. Panganiban, A. J. Pasuquin, M. J. Purganan, R. Malupeng, D. J. Pingad, J. P. Orolfo, H. H. Lua, and L. C. Velasco. Web traffic anomaly detection using isolation forest. *Informatics*, 11(4):83, 2024.

[2] A. Gupta, L. K. Tyagi, and A. Mohamed. A machine learning methodology for detecting sql injection attacks. In *Proceedings of the IEEE International Conference on Advances in Computing and Systems (ICTACS)*, pages 184–191, 2023.

[3] W. G. J. Halfond and A. Orso. Preventing sql injection attacks using amnesia. In *Proceedings of the 44th International Conference on Software Engineering*, 2006.

[4] M. Kazdagli, V. J. Reddi, and M. Tiwari. Quantifying and improving the efficiency of hardware-based mobile malware detectors. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[5] Y. Li, T. Zhang, Y. Yuan, MA, and C. Zhou. Anomaly detection of user behavior for database security audit based on ocsvm. In *2023 20th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pages 214–219, 2016.

[6] D. Lu, J. Fei, and L. Liu. A semantic learning-based sql injection attack detection technology. *Electronics*, 12(6):1344, 2023.

[7] Y. Nohara, K. Matsumoto, H. Soejima, and N. Nakashima. Explanation of machine learning models using shapley additive explanation and application for real data in hospital. *Computer Methods and Programs in Biomedicine*, 214:106584, 2021.

[8] V. Yepmo, G. Smits, M. Lesot, and O. Pivert. Leveraging an isolation forest to anomaly detection and data clustering. *Data & Knowledge Engineering*, 151:102302, 2024.