Adversarial Threats and Defense Mechanisms in Machine Learning-Based SQL Injection Detection: A Security Analysis

Ranwa Al Mallah
Royal Military College of Canada
Electrical and Computer Engineering
Kingston, Canada
Email: ranwa.al-mallah@rmc-cmr.ca

Alejandro Quintero
Polytechnique Montreal
Computer and Software Engineering
Québec, Canada
Email: alejandro.quintero@polymtl.ca

***Abstract*—SQL injection (SQLi) is a type of cyber attack where malicious code is inserted into a SQL query through an input field in a web application. This exploit targets vulnerabilities in the application's software that allows unsanitized or improperly validated user inputs to be executed as part of a database query. As a result, an attacker can gain unauthorized access to the database, retrieve, modify, or delete data, bypass authentication mechanisms, or perform other malicious actions. In Artificial Intelligence (AI), Machine Learning (ML) techniques are used to automatically detect SQL injections before they reach the web application. However, machine learning techniques are themselves vulnerable to attacks. We conduct an in-depth security analysis of a highly realistic threat model, experimentally demonstrating the effectiveness of poisoning and evasion attacks on a machine learning-based SQL injection detector. We then propose a defense mechanism tailored to the ML algorithm, and designed to protect the system against such attacks. This research highlights the consequences of inadequate management of AI technology in this field.**

I. Introduction

The range of attack vectors for web applications in the context of cybersecurity is vast. One of the most prevalent attack vectors is Structured Query Language (SQL) injections [1], [2], [3]. SQL injections are when users are able to modify input in such a way that they can perform all create, read, update, and delete (CRUD) operations on the target's SQL databases [4], [5], [6], [7].

The problem this study is trying to solve is to autonomously detect SQL injections before they reach the web application. This allows for the efficient alerting of network administrators to appropriately handle potential unauthorized access to data [8]. This can also be taken one step further, and the system can automatically block potential SQL injections without any user intervention. Having the ability to detect these injection attacks will ensure the confidentiality, integrity, and availability of web applications.

Classically, the approach to detecting SQL injections is to employ a signature-based algorithm. This means that the system has predefined sets of events that match SQL injections. The biggest limitation to this approach is that if the attack is not in the list of predefined signatures, it will pass undetected. This means that the system is useless against zero-day SQL injections [6], [9], [10].

The authors in [9] use various traditional ML techniques to solve the problem of SQL classification. The performance metric reported was the accuracy of 95.630% for the decision tree. Random forest increased accuracy to 96.525%; however, increased testing time of 30.5 seconds. When using a neural network, the accuracy was slightly higher of 0.2% higher.

The contributions of this paper can be summarized as follows:

- We will propose a robust binary classification machine learning model to predict if web application traffic is either a SQL injection or benign. Attacks will be conducted against this model and defenses will be implemented to validate robustness and enhance the security posture of the proposed model.
- A preexisting dataset will be used to train and test the SQL injection binary classification model using a supervised approach. Specific performance metrics will be considered because with SQL injection detection, false negatives are more detrimental than false positives. This is because if the model incorrectly labels a SQL injection as false, the injection attack will pass undetected potentially compromising the target's entire database. This is much worse than a false positive, where the worst that could happen is an administrator's time could be wasted. Because of this, it is more important that all SQL injections are detected rather than being more precise for the ones that are detected. As such, recall should be prioritized. For evaluating the performance of this model, recall will be prioritized over accuracy and precision. This means that the beta value for its f-score will be higher than 1.

The rest of the paper is organised as follows. In Section II we present the methodology where we construct the model in order to illustrate different attacks on it. In Section III, we provide the analysis of the results and suggest a defense to counter the attacks. We show the impact of the defense

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

against poisoning and evasion attacks. We conclude the paper and provide future work in Section IV.

II. Methodology

The dataset for detecting SQL injection is composed of collected SQL injection attacks and benign traffic from various web applications [11]. The dataset contains 33,726 unique samples, 22,305 benign samples and 11,456 samples are SQL injection attacks. Each sample contains the raw text of the traffic. As such, this will have to be converted into numerical values during the pre-processing stage.

The Deep Neural Network architecture used for classifying web traffic as a SQL injection attack or benign is a structure agnostic feedforward neural network. Manual testing was done to optimize the hyperparameters. The F2-score was the performance metric used. The following table provides a summary for the hyperparameters used.

TABLE I
HYPERPARAMETERS OF THE DNN.

Number of hidden layers	5
Number of neurons per hidden layer	1000-800-400-200-100
Dropout per hidden layer	50%-50%-40%-40%-0%
Number of neurons per output layer	1
Hidden layer activation function	ReLU
Output layer activation function	Sigmoid
Number of epochs	30
Minibatch size (samples per iteration)	80
Optimizer	adam

In Table II we provide a comparison of the performance of traditional machine models with DNN.

TABLE II
COMPARING DNN PERFORMANCE TO TRADITIONAL ML MODELS.

Model	**Accuracy**	**Precision**	**Recall**	**F2-Score**
DNN	93%	85.6%	97.4%	
Decision Tree	89.4%	78.1%	96.0%	
Naive Bayes	93.0%	85.6%	97.4%	
Random Forest	90.3%	80.0%	96.3%	

A. Data Poisoning Attack

The data poisoning attack implemented involves injecting static noise into benign and malicious samples to give the attacker a ”backdoor” for crafting SQL injections that will go undetected [12], [13]. This is especially effective for SQL injections where comments can be inserted as part of the attack. For example, if the static pattern injected into benign samples applies to the features for words ”america”, ”canada”, ”norway”, then the attacker can add these as a comment to their SQL injection query to avoid detection. For example, instead of doing

*; SELECT * FROM TargetTable*

they could do :

*; SELECT * FROM TargetTable #america canada norway*

to have the exact same query pass undetected.

One of the considerations for this kind of attack is to craft the static noise to arouse the least suspicion possible while maximizing the performance of the backdoor. At first, the pattern was applied to the five most occurring (most non-zero values) across all samples; however, this yielded relatively poor performance. As such, the static pattern is applied to the five least commonly occurring features (most zero values) across all samples. Another consideration is that if key features were targeted for the static injection, it would have a bigger impact on the performance, making it more likely for the attack to be detected. Fig.1 lists the least occurring feature values across all samples.

```
Values corresponding to bottom 5 features (to remove any that would arouse suspicion):
Word 'quot' Occurred 153 times across 1867 samples
Word 'apos' Occurred 147 times across 1867 samples
Word 'said' Occurred 99 times across 1867 samples
Word 'rownum' Occurred 59 times across 1867 samples
Word 'null' Occurred 54 times across 1867 samples
```

Fig. 1. Bottom Five Features Across all Samples.

From here, every unique non-zero feature is extracted from the training set and appended into an array. This is to ensure that the static patterns resemble other features to arouse the least suspicion possible. Five values are then chosen randomly from this for the benign static pattern and five values are chosen randomly for the malicious static pattern. Fig.2 shows an example of these patterns.

```
Static noise for malicious samples:  [0.2, 0.25, 0.3333333333333333, 0.25, 0.25]
Static noise for benign samples:  [0.2, 1.0, 0.3333333333333333, 0.5, 0.5]
```

Fig. 2. Generated Static and Benign Patterns.

These static injections are then applied to the least five occurring features for all the benign and malicious samples in the training set. The model is then trained with this static backdoor for malicious and benign samples. It should be noted that applying the poison to every malicious and benign sample is **not** indicative or a real-life scenario. This assumes that the attacker has access to the entire training set (white-box); however, in reality, the static pattern would only be applied to less than 5% of the samples.

Once the model has been poisoned, the attacker should be able to inject the benign pattern into a malicious sample to have it pass undetected (trigger a false negative) as well as inject the malicious pattern into a benign sample to have it pass undetected (trigger a false positive).

B. Adversarial examples

Adversarial examples are generated for the evasion attack [14], [15]. The Projected Gradient Descent (PGD) algorithm is used to create the adversarial examples for the evasion attack [16], [17]. This algorithm is essentially an iterative version of the Fast Gradient Sign Method (FGSM) algorithm. Instead of adding perturbations to the original sample, PGD initializes the sample to a random point within the Lp norm before adding the perturbation. The type of Lp norm is also a hyper parameter that can be set by the user. Also, PGD does random restarts throughout the process of finding the ideal perturbation (to maximize loss). This helps ensure that the algorithm does not

get caught in local maximums. Due to how expensive this random restart process is, it generally isn't used in practice. Table III shows the hyperparameters used in the PGD attack.

TABLE III
HYPERPARAMETERS USED IN THE PGD ATTACK.

Parameter	Value
Epsilon	0.01
Eps_iter (step size)	0.001
Iterations	10
norm (Lp norm)	Linf
clip_min (lowest feature val)	0
clip_max (highest feature val)	0.99

The clip_min and clip_max values are set to ensure that the values are between 0 and 0.99. This is to ensure they are reflective of the normalized values of the original input to arouse the least suspicion. Ideally, the values would also be rounded to their nearest neighbor; however, this did not yield as strong performance.

III. RESULTS AND ANALYSIS

Table IV shows the confusion matrix for the poisoning attack before the static pattern is injected into the test dataset.

TABLE IV
CONFUSION MATRIX FOR DNN **BEFORE** POISONING ON ORIGINAL TEST SET.

123 (TP)	37 (FP)
0 (FN)	170 (TN)

Table V shows the confusion matrix for the poisoning attack after the static pattern is injected into the test dataset.

TABLE V
CONFUSION MATRIX FOR DNN **AFTER** POISONING ON POISONED TEST SET.

43 (TP)	117 (FP)
77 (FN)	93 (TN)

Table VI shows the confusion matrix after adversarial samples have been injected into the test using the PGD algorithm.

TABLE VI
CONFUSION MATRIX FOR DNN AFTER PGD ADVERSARIAL EXAMPLES.

30 (TP)	130 (FP)
170 (FN)	0 (TN)

The performance of the DNN dropped significantly compared to the DNN under no attack.

For poisoning, this is expected since the static patterns were injected in such a way as to decrease performance and create a "backdoor" for attackers to intentionally misclassify samples [18].

This is also expected for evasion since the adversarial examples were constructed in such a way to maximize loss from calculating the gradients with respect to the inputs.

It should be noted that the evasion attack had a greater negative impact on performance than the poisoning attack. This is likely because the evasion attack was generated using an algorithm that is optimized for misclassification, whereas generic static noise was employed for the poisoning attack [19], [20], [21].

A. Defense against the attacks

A defense in depth strategy was implemented where three defense paradigms are used concurrently to detect the most SQL injections possible.

For the detection paradigm, an algorithm was written to detect adversarial example as well as poisoning on the training set. Before writing these algorithms, the training set was analyzed to determine what normal structure consists of. This includes: the most non-zero features included in a single sample, the unique non-zero features across all samples, the number of non-zero features that occur across all malicious samples and the number of non-zero features that occur across all benign samples. Fig. 3 shows the normal structure of the training set that was used to create the algorithms.

```
----Detecting what is normal on original training set-----
Sample with most non-zero features in training set:  17
Unique non-zero features:  [0.5, 1.0, 0.6000000000000001, 0.3333333333333333, 0.2
No non-zero feature appears across all malicious samples in training set
No non-zero feature appears across all benign samples in training set
```

Fig. 3. Normal Structure of Training Set.

The algorithm to detect poisoning determines whether static patterns exist across all benign and malicious samples. It loops through each feature of each sample to determine if the value is identical across all of them (excluding zeros). If more than two identical non-zero features appear across all malicious or benign samples, it is flagged as suspicious and the user is alerted of potential poisoning on the training set. This is more of an Intrusion Detection System approach where the user is just notified and nothing is automatically blocked. The idea is to have this defense run periodically on the training set to determine if poisoning has been done. It should also be noted that using auxiliary tools like this is considered the best defense for poisoning.

Unlike the poisoning detection algorithm which is used on the entire dataset, the evasion detection algorithm is used on individual samples [22], [23]. All features of the samples are analyzed and the number that do not appear in the unique values extracted from the training set are recorded. If this number exceeds a certain threshold, the sample is flagged as malicious. Furthermore, if the number of non-zero features across the sample exceeds 50, the sample is also flagged as malicious. The values used for these detection algorithms were based on what was seen in the training set with extra "padding" since the training set is not perfectly generalized to all practical web traffic.

From here, the data modification paradigm is used. This is done by training the model on adversarial examples that were generated using the same PGD algorithm and its corresponding

hyperparameters. Instead of creating the adversarial examples from the test set to train the model, adversarial examples were generated from the training set. It should be noted that even though the weights of the model will change with adversarial training, this still falls under the data modification paradigm. This is because the structure of the model remains consistent and all that is changing is the data that it is trained on.

Finally, the model modification paradigm is done through regularization. This is a similar idea to adversarial training where the model is trained with a certain perturbation; however, this noise is added to every single point during the training process and is part of the model's structure. A layer wise approach is used where the regularization term is added to every hidden layer and not just the last layer. This is identical to a parseval network; however, the regularization term is not added to the output layer (sigmoid activation function). Better performance was observed when the regularization terms were just applied at the hidden layers.

Regularization was employed using the L1, L2 and L1L2 regularization terms. L2 appeared to yield the highest results where the penalty term is the sum of the squared values of the weights. The default value of 0.01 was used for the term. Higher and lower values were tried; however, these seemed to cause drastic negative changes with respect to the performance on the original model.

For the detection paradigm, the poisoning attack was done on the training set and evaluated on the test set. In fact, the poisoning attack was done by injecting static patterns that consisted of previously existing values. These values are significantly larger than an appropriate normalization term. If the normalization term is increased to accommodate for these static patterns, the model vastly deteriorates. Because of this, the static patterns were scaled back so that the normalization terms would have an effect on the static pattern injection. If these static patterns exceeds these terms by a large enough margin, the detection paradigm will have to be relied upon to catch the static pattern injection. This is considered acceptable, since the detection paradigm is the best defense for poison (not including defense in depth) [24], [25]. Table VII shows the confusion matrix of the robust model after employing the static injection poisoning attack. Table VIII shows the resulting performance metrics.

TABLE VII
CONFUSION MATRIX FOR ROBUST DNN AFTER STATIC INJECTION POISONING.

128 (TP)	32 (FP)
0 (FN)	170 (TN)

For the detection paradigm, the evasion detection algorithm was run on an original sample, a sample generated using PGD and a sample generated using FGSM. Table IX shows the confusion matrix of the robust model after generating adversarial examples with the PGD algorithm and Table X shows the corresponding performance metrics.

TABLE VIII
SCALED PERFORMANCE OF ROBUST MODEL ON POISONED STATIC PATTERN INJECTION.

Accuracy	90.3%
Precision	80.0%
Recall	100%
F2-score	96.4%

TABLE IX
CONFUSION MATRIX FOR ROBUST DNN AFTER PGD ADVERSARIAL EXAMPLES.

137 (TP)	23 (FP)
0 (FN)	170 (TN)

TABLE X
SCALED PERFORMANCE OF ROBUST MODEL AFTER PGD ADVERSARIAL EXAMPLES.

Accuracy	93.0%
Precision	85.6%
Recall	100%
F2-score	97.4%

It should be noted that adversarial training was done using the PGD algorithm. This means that this type of defense would not have as much of an effect on adversarial examples generated with other algorithms (C&W for example).

The robust model performed well expect in the case of the static pattern injection attack when not scaled down. The static pattern had to be scaled down for the regularization term to have an effect. This scaled down static injection pattern was compared to the original and the robust model. This shows that the robust model is less affected by small perturbations; however, no difference could be seen when this perturbation exceeded a certain value. It should be noted that the poison detection algorithm still detected both static patterns highlighting that detection is the ideal defense for poisonning attcks. On the contrary, it should be noted that this detection algorithm would not work for practical examples where the pattern isn't injected in every benign/malicious sample. Table XI compares the original and robust DNN in the presence of the scaled down static pattern poisoning attack.

TABLE XI
COMPARING ORIGINAL AND ROBUST MODEL TO STATIC PATTERN POISONING, SCALED DOWN VERSION.

Model	**Accuracy**	**Precision**	**Recall**	**F2-Score**
Original DNN	77.3%	85.6%	72.5%	71.9%
Robust DNN	90.3%	80.0%	100%	96.4%

For the evasion attack, it could be seen that the robust model performed substantially better against PGD adversarial samples. This was mainly due to the model being trained on PGD adversarial samples during the training process. This would not have as much of an effect on adversarial samples generated with other algorithms (C&W, for example). It could also be seen that the detection algorithms caught all adversarial examples generated with PGD and FGSM. Table XII shows the

performance of the robust and original model in the presence of adversarial examples generated with the PGD algorithm.

TABLE XII
COMPARING ORIGINAL AND ROBUST MODEL ON PGD ADVERSARIAL EXAMPLES.

Model	Accuracy	Precision	Recall	F2-Score
Original DNN	8.5%	17.5%	14.1%	0%
Robust DNN	90.3%	80.0%	100%	96.4%

The performance of the robust model decreased slightly after training with the regularization terms and doing adversarial training. This is expected since if noise is injected during the training process, the model will perform better in the presence of similar noise; however, will perform worse when there is no noise. Due to the small decrease in the F2-Score (1%) and the vast increase of performance in the presence of attacks, it can be deduced that the robust model is more advantageous.

Finally, it should be emphasized that a DNN is not required for the use case of detecting SQL injections. It could be seen that the Naive Bayes algorithm had identical performance to the DNN. This is due to the simplicity of the patterns across SQL injections and how easily these can be picked up using prior probability. The Naive Bayes model should be used in practice due to the inherent security vulnerabilities present in DNNs.

Table XIII shows the performance of the original DNN compared to the performance of the robust DNN on the original test set.

TABLE XIII
COMPARING PERFORMANCE OF ROBUST DNN VERSUS ORIGINAL DNN ON TEST SET.

Model	Accuracy	Precision	Recall	F2-Score
Original DNN	93%	85.6%	100%	97.4%
Robust DNN	91.2%	82.5%	99.2%	96.4%

IV. CONCLUSION

This work explored the vulnerability of machine learning models to gradient-based adversarial attacks, which take advantage of the backpropagation algorithm. Neural networks, being overly linear, are particularly susceptible to adversarial perturbations of their inputs. In the context of SQL injection, improper use of AI technology can lead to serious consequences, from data manipulation to exploitation. Therefore, it is crucial to quantify and mitigate these risks. Adversarial robust training serves as an initial step in a broader mitigation strategy, helping to protect AI systems against input data perturbations.

REFERENCES

[1] M. Nasereddin, A. ALKhamaiseh, M. Qasaimeh, and R. Al-Qassas, "A systematic review of detection and prevention techniques of sql injection attacks," *Information Security Journal: A Global Perspective*, vol. 32, no. 4, pp. 252–265, 2023.

[2] A. Paul, V. Sharma, and O. Olukoya, "Sql injection attack: Detection, prioritization & prevention," *Journal of Information Security and Applications*, vol. 85, p. 103871, 2024.

[3] V. Abdullayev and A. S. Chauhan, "Sql injection attack: Quick view," *Mesopotamian Journal of CyberSecurity*, vol. 2023, pp. 30–34, 2023.

[4] D. Alghazzawi, M. Algawazi, and S. Alarifi, "Detection of sql injection attack using machine learning techniques." [Online]. Available: https://www.mdpi.com/2624-800X/2/4/39

[5] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[6] J. Clarke, "Sql injection attacks and defense," pp. 1–2, 2012. [Online]. Available: https://books.google.ca

[7] J. R. Tadhani, V. Vekariya, V. Sorathiya, S. Alshathri, and W. El-Shafai, "Securing web applications against xss and sqli attacks using a novel deep learning approach," *Scientific Reports*, vol. 14, no. 1, p. 1803, 2024.

[8] H. Holm, "Signature based intrusion detection for zero-day attacks," 2014. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6759203

[9] K. Ross, "Sql injection detection using machine learning techniques and multiple data sources," 2018. [Online]. Available: https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1649

[10] S. H. Shan. (2023) Mr. shah's linkedin. [Online]. Available: https://www.linkedin.com/in/syed-saqlain-hussain-shah-5494b8b6/?originalSubdomain=pk

[11] S. Shah. (2021) sql injection dataset. [Online]. Available: https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset?select=sqliv2.csv

[12] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[13] A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli, "Wild patterns reloaded: A survey of machine learning security against training data poisoning," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–39, 2023.

[14] R. Sagar, R. Jhaveri, and C. Borrego, "Applications in security and evasions in machine learning: a survey," *Electronics*, vol. 9, no. 1, p. 97, 2020.

[15] M. Macas, C. Wu, and W. Fuertes, "Adversarial examples: A survey of attacks and defenses in deep learning-enabled cybersecurity systems," *Expert Systems with Applications*, vol. 238, p. 122223, 2024.

[16] A. Aldahdooh, W. Hamidouche, S. A. Fezza, and O. Déforges, "Adversarial example detection for dnn models: A review and experimental comparison," *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4403–4462, 2022.

[17] L. Chen, Y. Zhang, Y. Song, L. Liu, and J. Wang, "Self-supervised learning of adversarial example: Towards good generalizations for deepfake detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 18 710–18 719.

[18] R. Al Mallah, B. Lachine, and G. Badu-Marfo, "Security of sdn in an adversarial setting: The ddos case," in *2023 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*. IEEE, 2023, pp. 53–58.

[19] skikit-learn. [Online]. Available: https://scikit-learn.org/

[20] H. Sachdev. (2020) Choosing number of hidden layers and number of hidden neurons in neural networks.

[21] pandas. [Online]. Available: https://pandas.pydata.org/

[22] M. A. Ayub, W. A. Johnson, D. A. Talbert, and A. Siraj, "Model evasion attack on intrusion detection systems using adversarial machine learning," in *2020 54th annual conference on information sciences and systems (CISS)*. IEEE, 2020, pp. 1–6.

[23] G. Fidel, R. Bitton, and A. Shabtai, "When explainability meets adversarial learning: Detecting adversarial examples using shap signatures," in *2020 international joint conference on neural networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[24] M. Tayyab, M. Marjani, N. Jhanjhi, I. A. T. Hashem, R. S. A. Usmani, and F. Qamar, "A comprehensive review on deep learning algorithms: Security and privacy issues," *Computers & Security*, vol. 131, p. 103297, 2023.

[25] A. Takiddin, M. Ismail, U. Zafar, and E. Serpedin, "Robust electricity theft detection against data poisoning attacks in smart grids," *IEEE Transactions on Smart Grid*, vol. 12, no. 3, pp. 2675–2684, 2020.