Detection of SQL Injection Vulnerability using Q-Learning Reinforcement Learning Agents

Tejas Sheth[1], Janhavi Anap[2], Het Patel[3], Nidhi Singh[4], and Prof. Ramya R B[5]

[1] Student, Dept. of Computer Engineering, A. P. Shah Institute of Technology, Thane
tejas.sheth04@gmail.com
[2] Student, Dept. of Computer Engineering, A. P. Shah Institute of Technology, Thane
anapjanhavi@gmail.com
[3] Student, Dept. of Computer Engineering, A. P. Shah Institute of Technology, Thane
hetpokar@gmail.com
[4] Student, Dept. of Computer Engineering, A. P. Shah Institute of Technology, Thane
singhnidhik2002@gmail.com
[5] Guide, Dept. of Computer Engineering, A. P. Shah Institute of Technology, Thane
ramyarb@apsit.edu.in

Abstract. The SQL Injection vulnerability can either be unintentional by software developers or an intentional ploy employed by a hacker with malicious intent to exploit sensitive data. With the recent surge in information, there is an innate quest to safeguard this information from falling into the wrong hand leading to data theft, leak of personal data or loss of property. With relational databases like MySQL being the most popular, it allows users to extract any available information without any significant knowledge of databases. With vast information stored in databases warrants attacker's attention, potentially risking critical confidential information. The premature detection of SQL Injection Attacks will be very helpful in preventing any malicious attempt by an attacker. In this research, we analyze the results of Reinforcement Learning algorithms like Q-Learning on a dataset consisting of potential SQL Injection queries. We intend to provide a Reinforcement Learning solution to minimise the potential threat posed by SQL Injection and give apriori to the model to learn to detect a SQL attack and prevent any unforeseen mishap more quickly and accurately.

Keywords: Machine Learning Information security SQL injection SQL detection SQL injection Attacks Reinforcement Learning Q-Learning Vulnerability detection Apriori

1 Introduction

Cybersecurity is the method of defending systems and programs from digital strikes. These strikes are usually designed for retrieving, manipulating or destroying sensitive data or interrupting standard business approaches. Cybersecurity

aims to reduce the risk of cyber-attacks and to protect against the illegal manipulation of networks and systems. Implementing robust and reliable measures is challenging due to the ever-evolving innovative attackers.

One of the most noxious security exploits widely used by hackers is Structured Query Language Injection Attacks (SQLiA) according to Open Web Application Security Project (OWASP) Top 10 report 2022 [1]. SQL Injection (SQLi) is an approach that hackers use to acquire an illegitimate entryway to a database with the use of malicious input to a database request.

SQL is a computer programming language, used to create, retrieve, update and delete data stored in relational databases. SQL is a standard database language, which is widely used in all relational database management systems (RDBMS) like MySQL. SQL is advantageous in permitting users to access data, describe data, define data and manipulate data. It also authorizes users to create databases, create and drop tables, create views, stored procedures and functions, and also set permissions on tables, views and procedures. SQL allows embedding using modules, libraries and pre-compilers with other languages.

The main reason why SQLi keeps traction is due to improper and insecure development architecture. Making use of legacy software introduces security vulnerabilities that might not be detected with modern software. Using authorized and the latest versions of the software are critical to avoiding or minimizing potential security exploits, including SQLiA.

SQLiA mainly occurs through the concatenation of a malicious SQL command at the end of a given input query on the user frontend. In recent times, there has been an advancement in the number of dynamic websites, that rely on relational databases to store and manipulate large amounts of data in order to provide a rich user experience. One of the most common operations of SQL is Data Retrieval using the SELECT command coupled with the WHERE clause.

A few of the common SQLiA types are Piggy-Backed, Stored Procedures, Union Query, and Alternative Encoding according to Halfond, W. G. et. al. [2] and Wei, K. et. al. [3]. There are some advanced SQLiA types like Blind SQLi where the attackers devise a strategy to circumvent the lack of error notifications, Fast Flux SQLi where the DNS method is used to conceal phishing and malware distribution sites behind a constantly changing network of compromised servers, Compounded SQLi is where the attacker uses a combination of two or more seizures that target the web page and it has far-fetching implications.

The detection of these attack types becomes a very important task keeping in mind the potential data breaches that could happen. The detection methods range from checking server logs to monitoring database errors. The pattern match method is a combination of static analysis and run-time monitoring to detect SQL Injection attacks still remain the most common.

Another approach to detecting SQLiA is by making use of various Machine Learning techniques like supervised and unsupervised, Reinforcement Learning. There are many supervised and unsupervised approaches towards detecting SQLiA.

In this paper, we have used the Reinforcement Learning method towards SQLiA detection. Here, in the training phase, we provide apriori to the Q-Learning model. The training dataset contains both malicious and non-malicious queries. Providing efficient apriori is the most crucial factor in order to obtain higher efficiency. The advantage of Reinforcement Learning techniques over others is that it allows us to explore a variety of SQL queries and the accuracy keeps increasing with the increase in the use of the model. Also, owing to Reinforcement Learning methodologies, the accuracy value of SQL detection is raised and the false positive rate drops.

2 Literature Survey

A malicious SQL query is inserted into a web application by combining it with the input parameters, known as SQL Injection. Sadeghian, A. et. al. illustrate the classification of injection attacks like tautology queries, illegal or logically incorrect queries, union queries, piggy-backed queries, stored procedures, inference, and alternate encodings [4]. Security researchers have categorized the solutions for SQLi into three major categories: Best code practices, SQLi detection and SQLi runtime prevention. The optimum solution would be writing secure code and among best code practices- parameterized querying is the most secure and efficient technique.

Rai, A. et. al. illustrate the classification and prevention of different SQLi attacks. SQLi is generally classified as In-band SQLi, Inferential SQLi and Out of Bound SQLi [5]. In-Bound SQL injection is further classified as Error-based and Union-based SQLi. Inferential SQLi can be broken down into Boolean-based Blind SQL and Time-based SQL. Defensive techniques that could be used to prevent an SQLi attack include Whitelisting/Blacklisting, prepared statement/ parameterized query, stored procedure, defensive coding practice, taint-based approach, proxy filters, instruction set randomization, low privileges and output Escaping. Different countermeasures work for different SQL Attacks.

Medhane and M. H. A. S. based their approach on grammar to determine the injection vulnerabilities during software development and SQLi attack based on web applications [6]. By employing SQL queries, the attackers were able to execute their assault, causing a modification in the program's behavior as the SQL queries were modified, thus disrupting its intended function.

John, A. contemplated methods incorporating the best qualities of parse tree validation and code conversion techniques [7]. The algorithm parses the user input and checks for vulnerability, and if found, it applies code conversion over that input. Results show few drawbacks of code conversion as applying it to every user input is labour-intensive as well as the database increases. The parse tree validation technique could raise a false alarm if a legitimate user is having blank space in their input. The proposed method proved to provide higher security levels than the individual techniques of code conversion and parse tree validation.

Hanmanthu, B. et. al. illustrates the use of the famous decision tree classification techniques to prevent SQLi attacks [8]. The considered model works by sending different particularly planned attack requests to the proposed SQLi decision tree model, and the final SQLi database is created for using classification data. It uses the satisfied analysis technique for finding the SQLi attack and uses the SQL decision tree. Typically, software developers employ string concatenation to dynamically build SQL statements. However, this approach can be prone to errors and may require manual intervention. The proposed method allows for the creation of multiple queries tailored to meet the different conditions specified by users. This approach eliminates the need for manual intervention and mitigates the risks associated with error-prone coding. The model showed consistency in attack detection and elimination at an average of 82% for all types of attacks.

Tang, P. et.al. only extracted and classified the URL features [9]. The factors like payload length, keywords and their weights are considered for feature extraction. The URL is classified as malicious or non-malicious using Artificial Neural Network (ANN) models. The method and algorithm used here are multi-layer perceptrons (MLP) and LSTM, both implemented using Pytorch. The trained model is deployed in the ISP system so that abnormal behaviours can be found in the network in real time. While using the LSTM model for recognition purposes in this approach, there were some drawbacks. These include reduced accuracy, poor model recognition, and increased processing time, which can be problematic for the overall performance of the system.

Reinforcement Learning (RL) is known to obtain knowledge by trial-and-error method and continuous interaction with a dynamic environment. It is characterized by self-improving and online learning, making it one of the intelligent agents (IA) core technologies. TIn reinforcement learning (RL), the signal provided by the environment serves as a form of evaluation for the quality of the actions taken by the AI. However, this signal does not provide instructions on how to generate the correct action. The basic model of RL as stated in Qiang, W., & Zhongli, Z. includes a state, action and reward system [10]. Where the IA perceives the environment and chooses an action to obtain the biggest reward value by continuously interacting with the environment. The ultimate goal of RL is to learn an action strategy. At the core of reinforcement learning technology lies the idea that a system's actions which yield positive rewards from the environment will reinforce its tendency to produce similar actions in the future, thereby creating a positive feedback loop. Conversely, actions that do not produce a positive reward will weaken the system's tendency to generate similar actions. Typical RL method based on the Markov decision-making process (MDP) model includes two kinds: Model-based methods such as the SARSA algorithm and Model-irrelevant methods, like the Q-Learning algorithm.

Ghanem M. C., & Chen T. M. propose and evaluates an AI-based pen-testing system which makes use of RL to learn and replicate standard and complicated pen-testing activities [11]. The scope is limited to network infrastructure PT planning and not the entire practice. Moreover, the authors tackle the tricky

problem of expertise capturing by allowing the learning module to store and reuse PT policies in a more efficient way.

Niculae, S. et al. measured the performance of multiple fixed-strategy and learning-based agents [12]. They concluded that Q-Learning, with some extra techniques applied and greedy agent initialisation, performed best, surpassing human performance in the given environment.

Hu, Z., Beuran, R., & Tan, Y. suggest an automated penetration testing framework, based on deep learning techniques, particularly Deep Q-Learning networks (DQN) [13]. The authors conducted an experiment in which a given network host was populated with real host and vulnerable data, to determine the optimal attack path, and to provide viable solutions.

Erdődi, L. et. al. simplified the dynamics of SQLi vulnerabilities by projecting the problem as capture-the-flag security and implementing it as an RL problem [14]. Assuming the vulnerability has been recognized, they rely on RL algorithms to automate the process of exploiting SQLi. They implemented the model using two simulations. The first simulation showed that a simple RL agent-based Q-Learning algorithm can successfully develop an effective strategy to resolve the SQLi problem. Through pure trial and error, a tabular Q-Learning algorithm can uncover an effective strategy and achieve performance levels close to the theoretical optimum. However, while the use of a table to store the Q-value function enables a detailed analysis of the agent's learning dynamics, this approach suffers from poor scalability, limiting its practical applicability. Thus, in the second simulation, they sacrificed interpretability to work around the issue of scalability. A deep Q-Learning agent was deployed to tackle the problem in the first simulation, which was able to learn a good strategy for the SQLi problem, and also provide a solution to the space constraints imposed by the instantiation of an explicit Q-Table.

RL is fairly successful in tackling and solving games, penetration testing, when distilled as a capture-the-flag (CTF), can be expressed as a game. When it comes to penetration testing, a machine may be limited to learning by trial and error, whereas a human hacker can utilize other methods such as knowledge from alternative sources, deduction, hypothesis testing, and social engineering. Although an RL agent may in principle learn the structure from scratch in a pure model-free way, this may be computationally difficult. Thus according to Zennaro, F. M., & Erdodi, L. injecting some form of elementary apriori knowledge about the structure of the problem may simplify the learning problem [15]. Some basic forms of apriori knowledge are lazy loading, state aggregation and imitation learning, which makes the RL agent much more efficient. The authors categorized CTFs in groups according to the type of vulnerability they instantiate and the type of exploitation that a player is expected to perform. The prototypical classes of CTF problems considered were port scanning and intrusion, server hacking and website hacking. All the simulations were implemented using the standard RL interface defined in the OpenAI gym library.

Verme, M. D. et. al. contemplated the issue of exploiting SQLi vulnerabilities and representing it as a capture-the-flag scenario in which an attacker can

submit strings to an input form with the goal of acquiring a flag token representing private information [16]. The attacker was modelled as an RL agent that maintains interaction with the server to learn an optimal policy leading to the attacker taking advantage of it. The authors did a comparison between two types of agents, one was a simple structured agent that relied on significant apriori knowledge and used high-level actions while the other was a structureless agent that had limited apriori knowledge and generated SQL statements. The comparison showcased the feasibility of developing agents that relied on less ad-hoc modelling.

3 Reinforcement Learning

Reinforcement Learning is a type of Machine Learning where an agent learns to make decisions by interacting with an environment. Here there are unlabeled data, so the agent learns through experience. The agent interacts with the environment so as to maximize a reward signal. The agent learns the concept of a reward-based mechanism using the hit-and-trial method. It is used in an application, such as game-playing, robotics and decision-making.

3.1 Components of Reinforcement Learning

Agent- entity that interacts with the environment and makes decisions based on the feedback it receives. **Environment**- a world from which the agent operates and receives feedback. **State**- represents the environment at a given time, which the agent uses to make decisions. **State Space**- set of all possible states the agent could take to achieve the required goal. **Action**- decision that the agent makes based on the current state of the environment. **Action Space**- a Finite set of possible actions that the agent can take. Reward- is an evaluation parameter returned from the environment to the agent. **Policy**- strategy applied by an agent for the next action based on the current state. **Value**- compared to the short-term reward, is the expected long-term return with a discount. **Q-Value**- similar to the value, but it takes one additional parameter known as current action (a). It is the expected cumulative reward of taking an action in a particular state and following a particular policy.

3.2 Reinforcement Learning Algorithms

Q-Learning - Q-Learning is a model-free reinforcement learning algorithm. It is also known for its ability to capture complex relationships within input features and to handle non-stationary environments where the underlying distribution of the data may change over time. In this research, temporal difference learning has been used. It is an algorithm that chooses the best action that should be taken at any possible state by estimating the value of each action known as a value-based algorithm. In Q-Learning, the agent maintains a Q-table which contains the estimated Q-values for each state-action pair. On each step, the agent selects

an action to be performed based on its current state and updates the Q-table based on the observed reward and the estimated Q-values of the next state. Over time, as the agent experiences more states and updates its Q-table, the estimated Q-values converge to the true optimal Q-values, permitting the agent to learn the optimal policy. Thus, the algorithm guarantees to obtain optimal solutions.

Q-Learning guarantees global convergence to an optimal policy under certain conditions, which can provide more confidence in the learned policy than SARSA or DQL. Thus making it a more reliable option for SQLi detection, where the consequences of false positives and false negatives can be severe.

State Action Reward State action (SARSA) - The one major variance between Q-Learning and SARSA algorithms is that, unlike Q-Learning, the maximum reward for the next state is not required for updating the Q-value in the table. The action that the agent takes would be to either label the query as legitimate or malicious. The reward signal would be based on the accuracy of the agent's labelling. During training, the agent uses SARSA to learn a policy that maximizes the expected future reward. The agent starts in an initial state and selects an action based on its policy. It then observes the reward and the new state resulting from the action and updates its Q-value based on the SARSA update rule. Once the agent has learned a policy through training, it can be used to detect SQL injection attacks in real time. The agent would observe a new SQL query and use its policy to label the query as legitimate or malicious.

Compared to Q-learning, SARSA is an on-policy algorithm, which means it learns a policy while taking into account the current policy being used. This can be useful in situations where the agent needs to balance exploration and exploitation in a more controlled manner, and where the reward signal may be noisy or delayed. Unfortunately, SARSA is known to converge slower than the Q-learning algorithm which can be an issue when training the algorithm on large datasets of SQL queries. SARSA has several hyperparameters that need to be carefully tuned to achieve optimal performance. This can be a time-consuming and challenging task, especially when dealing with large datasets.

Deep Q-Learning - It is a combination of reinforcement learning with deep neural networks, which allows for more complex representations of the state and action spaces. Deep Q-Learning (DQL) is a variant of Q-Learning that uses a neural network to approximate the Q-function, rather than a lookup table. During training, the agent uses DQL to learn a policy that maximizes the expected future reward. The agent observes a state and passes it through a neural network to estimate the Q-values for each action i.e., legitimate or malicious. The agent selects the action with the highest Q-value based on an epsilon-greedy policy. The agent then observes the reward and the new state resulting from the action and updates the neural network using the back-propagation algorithm. Once the agent has learned a policy through training, it can be used to detect SQL injection attacks in real time. The agent observes a new SQL query, passes

it through the neural network to estimate the Q-values for each action, and selects the action with the highest Q-value. The agent then labels the query as legitimate or malicious on the basis of the selected action.

Compared to traditional Q-Learning, DQL can handle high-dimensional state spaces more efficiently by using a neural network to approximate the Q-function. However, DQL requires a large amount of computation, particularly when using large neural networks. This can be a significant disadvantage when training the algorithm on large datasets of SQL queries. DQL can also be susceptible to overfitting, particularly when the dataset is small or noisy. Overfitting can lead to poor generalization to new SQL queries, reducing the algorithm's ability to detect SQL injection attacks.

3.3 Implementation of Q-Learning

Algorithm - The Q-learning algorithm works by building a table of Q-values that represent the expected rewards for taking a particular action in a given state. The Q-value for a particular state-action pair represents the expected reward that can be achieved by taking that action in that state and following the optimal policy thereafter. The algorithm starts by initializing the Q-table with arbitrary values, and then iteratively updates the Q-values based on the rewards received during each trial. An action a is taken based on the current state and Q-table. The Q-learning algorithm continues to iterate until the Q-values converge to the optimal values. The optimal policy can be obtained by selecting the action with the highest Q-value for each state.

Use of Apriori - In Reinforcement Learning, the Apriori algorithm can be used to identify patterns in the interactions between an agent and its environment. They help the agent optimize its decision-making and improve its overall performance. If the agent frequently takes a certain action when it is in a specific state, the Apriori algorithm identifies this pattern and suggests that the agent take this action in the future whenever it is in a similar state. This helps the agent avoid sub-optimal decisions and improve its overall reward.

In our model, we provided CNN model as an apriori to the Q-learning algorithm. The CNN model was chosen over any other model like Naive Bayes because of its better ability to recognize a pattern in the input data that indicates the presence of an SQL injection attack. The variation in SQL injections is subtle, such as changes in the order of parameters or the use of different operators. CNN models are robust to these variations, as they can learn to recognize patterns regardless of their location in the input data. CNN models also have the ability to learn complex patterns that are composed of simpler patterns. And most importantly CNN models are computationally efficient and can be trained on large datasets. This is important for SQL injection detection, as the model needs to be trained on a large and diverse set of input data in order to generalize well to new and unseen attacks.

Apriori can be integrated into a Q-Learning model at various stages of the learning process depending on the requirements. Apriori can be integrated at

the pre-processing stage when identifying frequent patterns or correlations in the data and transforming the data into a more suitable format for the Q-Learning algorithm is crucial. Apriori can be used in the exploration/exploitation trade-off stage to balance the need for exploration with the need for exploitation and ensure that the Q-Learning algorithm is able to learn effectively from the available data. Apriori can also be integrated into the Q-Value update stage where certain state-action pairs are prioritized over others to improve the accuracy and efficiency of the Q-learning algorithm.

Thus, we will be integrating the Apriori of the CNN model in the Q-Value update stage. The CNN model will provide robustness while providing apriori at the Q-value update stage, which will improve the accuracy. In a Q-Learning environment model, a step function is a function that defines the behaviour of an agent as it interacts with its environment. The step function is used to update the Q-Value estimates for each state-action pair based on the observed reward and the predicted reward for the next state. Integrating the Apriori algorithm into the step function of a Q-Learning model involves using the interpretations of the apriori (CNN) model to modify the Q-Value estimates and the algorithm of the CNN model added as apriori to the step function of a Q-Learning environment model remains similar to the standard algorithm of Q-Learning model. The only difference is in the definition of the step function.

To integrate apriori in the Q-Learning model,

1. Define the Q-learning environment model and its functions of it like reset and step. Also, define the Q-learning agent model and its function like act.
2. The step function will take the query as input and with the help of the apriori of the CNN model it will determine the reward.
3. The act function will make the prediction based on the policy the Q-Learning model has created.

Updating of Q-Table - The Q-value for a state-action pair (s, a) is updated using the observed reward after taking action a in state s and the predicted future rewards. The update is done using the following equation: $Q(s,a) = Q(s,a) + \alpha * (r + \gamma * max(Q(s',a')) - Q(s,a))$

where, Q(s, a) is the current estimate of Q-value for state-action pair (s, a).

α is the learning rate, which determines the extent to which new information overrides the old estimate of the Q-value.

r is the reward observed after taking action a in state s.

γ is the discount factor, which determines the importance of future rewards relative to the present reward.s

Q(s', a') is the estimated Q-value for the next state s' and the best action a' in that state.

4 Experimental Outcomes

The learning rate is a trade-off between the algorithm's stability and its ability to respond to changes in the environment. It is a scalar value that ranges between 0

and 1. If the learning rate is high, the agent quickly assimilates new information and responds rapidly to the changes in the environment. However, a high learning rate makes the algorithm more unstable, since the agent overreacts to noisy information as seen in Fig. 1. If the learning rate is low, the agent updates its estimates more slowly and is less sensitive to changes in the environment. However, a low learning rate can also make the algorithm slow to meet to the optimal policy, since it takes longer for the agent to learn from its experiences.

The discount factor influences the Q-Learning algorithm's balance between short-term and long-term rewards. It is a scalar value that ranges between 0 and 1. A high discount factor (close to 1) means that future rewards are considered to be very important, and the agent will prioritize them over immediate rewards. A low discount factor (close to 0) means that the agent only cares about immediate rewards and does not consider future rewards as seen in Fig. 2.

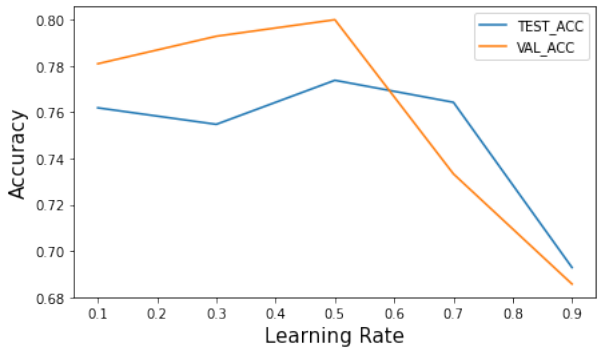

Fig. 1. Effect of Learning Rate on Accuracy

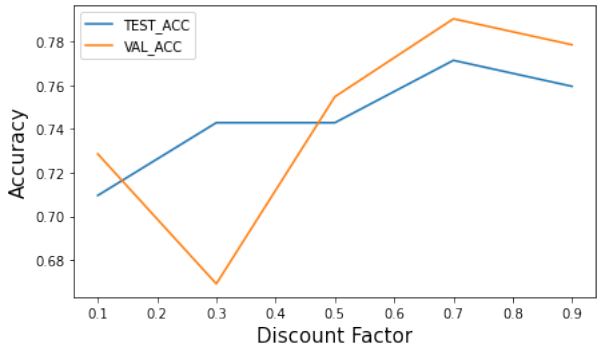

Fig. 2. Effect of Discount Factor on Accuracy

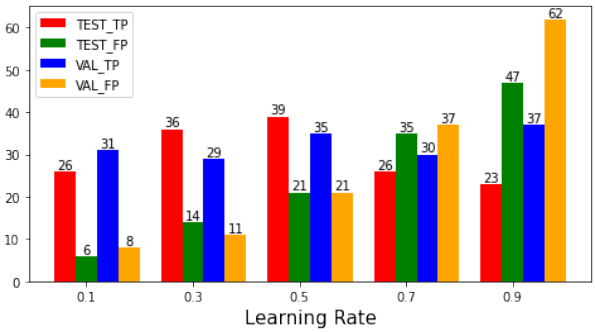

Fig. 3. Effect of Learning Rate on Testing and Validation Predictions

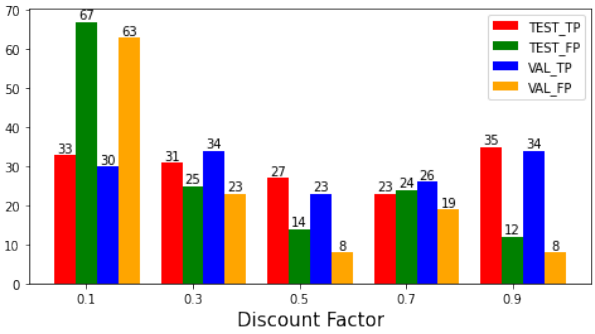

Fig. 4. Effect of Discount Factor on Testing and Validation Predictions

5 Conclusion

Using a Q-Learning algorithm for detecting SQL injection attacks in Web Applications holds promise in achieving accurate and real-time detection while minimizing false positives and negatives. The algorithm can be trained on a dataset of SQL queries and their labels to learn a policy for detecting malicious queries. Further research in this area can lead to the development of more sophisticated and robust algorithms for securing web applications against SQL injection attacks. To increase the accuracy of the Q-Learning model, it is suggested to use a learning rate closer to 0 and a discount factor closer to 1 as seen from Fig. 1 and Fig. 2 where the accuracy for validation data is greater than the accuracy for testing data in an ideal condition. This can increase the accuracy from 66% to 76%. As the model learns from the training data, further learning with testing and validation data can lead to an increased accuracy from 70% to 76%. With

a learning rate of 0.1 and discount factor of 0.9, not only can better accuracy values be obtained, but the difference between the number of actually detected injections and false alarms can also be maximized as seen from Fig. 3 and Fig. 4.

References

1. "OWASP Top Ten", https://owasp.org/www-project-top-ten
2. Halfond, W. G., Viegas, J., and Orso, A. A Classification of SQL-Injection Attacks and Countermeasures. In SSSE (2006).
3. Wei, K., Muthuprasanna, M., & Suraj Kothari. (2006). Preventing SQL injection attacks in stored procedures. Australian Software Engineering Conference (ASWEC'06).
4. Sadeghian, A., Zamani, M., & Abdullah, S. M. (2013). A Taxonomy of SQL Injection Attacks. 2013 International Conference on Informatics and Creative Multimedia.
5. Rai, A., Miraz, M. M. I., Das, D., Kaur, H., & Swati. (2021). SQL Injection: Classification and Prevention. 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM).
6. Medhane, M. H. A. S. (2013). Efficient solution for SQL injection attack detection and prevention. International Journal of Soft Computing and Engineering (IJSCE), 3, 396-398.
7. John, A. (2015). SQL Injection Prevention by adaptive algorithm. IOSR journal of computer engineering, 17, 19-24.
8. Hanmanthu, B., Ram, B. R., & Niranjan, P. (2015). SQL Injection Attack prevention based on decision tree classification. 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO).
9. Tang, P., Qiu, W., Huang, Z., Lian, H., & Liu, G. (2020). Detection of SQL injection based on artificial neural network. Knowledge-Based Systems, 105528. doi:10.1016/j.knosys.2020.105528
10. Qiang, W., & Zhongli, Z. (2011). Reinforcement learning model, algorithms and its application. 2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC).
11. Ghanem, M. C., & Chen, T. M. (2019). Reinforcement learning for efficient network penetration testing. Information, 11(1), 6.
12. Niculae, S., Dichiu, D., Yang, K., & Bäck, T. (2020). Automating penetration testing using reinforcement learning.
13. Hu, Z., Beuran, R., & Tan, Y. (2020). Automated Penetration Testing Using Deep Reinforcement Learning. 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW).
14. Erdődi, L., Sommervoll, Å. Å., & Zennaro, F. M. (2021). Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents. Journal of Information Security and Applications, 61, 102903.
15. Zennaro, F. M., & Erdodi, L. (2020). Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge. arXiv preprint arXiv:2005.12632.
16. Verme, M. D., Sommervoll, Å. Å., Erdődi, L., Totaro, S., & Zennaro, F. M. (2021, November). SQL Injections and Reinforcement Learning: An Empirical Evaluation of the Role of Action Structure. In Nordic Conference on Secure IT Systems (pp. 95-113). Springer, Cham.