LITERATURE REVIEW:

SQL Injection (SQLi) is a type of attack in which an attacker inserts a malicious SQL query into the web application by appending it to the input parameters. Reference [1] illustrates the classification of injection attacks like tautologies, illegal / logically incorrect queries, union queries, piggy-backed queries, stored procedures, inference, and alternate encodings. Security researchers have categorized the solutions for SQLi into three main groups: 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.

Reference [2] illustrates the classification and prevention of different SQLi attacks. SQLi is generally classified as In-band SQLi, Inferential SQLi and Out of Bound SQLi. 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.

The author in [3] based their approach on SQLi grammar to identify the SQLi vulnerabilities during software development and SQLi attack based on web-based applications. The attacker’s area unit used SQL queries for assaultive and hence these attacks reshare the SQL queries, thus neutering the behaviour of the program.

Reference [4] proposed methods which consist of the best features of parse tree validation technique and code conversion technique method. The algorithm parses the user input and checks whether it's vulnerable if any chance of vulnerability is found it applies code conversion over that input. Results show few drawbacks of code conversion as applying it to every user input is more time consuming and as well as the database also increases. The parse tree validation technique could raise a false alarm if a legitimate user is having blank space in his/her input. The proposed method proved to provide higher security levels than the individual techniques of code conversion and parse tree validation.

Reference [5] illustrates the use of the famous decision tree classification techniques to prevent SQLi attacks. The proposed model works by sending different specially 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. Software engineers usually rely on dynamic query building with string concatenation to construct SQL statements. The proposed method makes it possible to engineer different queries based on varying conditions set by users, without the need for manual interactions or error-prone code. The model showed consistency in attack detection and elimination at an average of 82% for all types of attacks. In order to perform a comparative evaluation of the proposed model, [5] compared the proposed model to the other SQL scanning model which includes Acuneits, Netsparker, and Web cruiser and the results of the proposed model show good accuracy in comparison to other models.

Reference [6] gives us an idea about different types of SQLi attacks as already mentioned by reference [2]. The three main types are Classic In-band SQLI, Inferential Blind SQLI, and SQLi Based On Out-of-Band. They present the comparative analysis of different supervised ML algorithms to mitigate SQLi attacks. Besides precise accuracy and minimum errors, ML models also require putting several factors into consideration. The following metrics were taken into consideration to decide the effectiveness of the algorithm: Kappa Statistic, True Positive (TP) Rate, Accuracy, True Negative (TN) and (time to build the model (TTB)), for each of the machine learning algorithms.

Reference [7] only extracts and classifies the URL features. The factors like payload length, keywords and their weights are considered for feature extraction. The URL is classified as malicious or non-malicious using ANN (Artificial Neural Network) models. The method and algorithm used here are multi-layer perceptron (MLP) and LSTM, both of which were 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. One of the drawbacks of using such an approach is that using the LSTM, model recognition is poor with high processing time & has lower accuracy.

Four machine learning models were considered in reference [8] and they were compared, SupportVector Machine (SVM), Boosted Decision Tree, Artificial Neural Network, and Decision Tree. They have proposed a framework using compiler platform and ML to detect SQLi in queries which are illegal and logically incorrect on server-side scripting. The dataset consists of 1100 samples of vulnerable SQL commands. After training the model with the dataset it was evaluated in terms of probability of detection, probability of false alarm, precision, accuracy, and processing time. Decision Jungle was the best in performance showing results as the best machine learning model which related to the processing time of 2.4725 seconds and accuracy of 0.9968.

Reference [9] collects traffic from two points: a web application host and a Datiphy appliance node. It is demonstrated that the accuracy obtained with correlated datasets using algorithms such as rule-based and decision-tree are nearly the same as those with a neural network algorithm, albeit with significantly improved performance.

Reinforcement Learning (RL) is known for obtaining knowledge by trial and error and continuously interacting with a dynamic environment. It is characterised by self-improving and online learning, making it one of the intelligent agents (IA) core technologies. The reinforcement signal provided by the environment in RL is to make a kind of appraisal of the action quality of the IA, but not tell the IA how to generate the correct action. The basic model of RL as stated in reference [10] includes a state, action and reward system. 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. The basic theory of reinforcement learning technology is: If a certain system's action causes a positive reward for the environment, the system generating this action lately will strengthen the trend, this is a positive feedback process; otherwise, the system generating this action will diminish this trend. 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, such as the TD algorithm and the Q-learning algorithm.

Reference [11] illustrates methods to generate more effective penetration test case inputs to detect SQLi vulnerability. The model-based penetration test

method is found to generate test cases covering more types and patterns of SQLi attack input to thoroughly test the ‘blacklist filter mechanism’ of web applications. Here, the authors proposed two-step penetration test case generation, building and instantiating, where step 1 reveals what test case should be used while step 2 expounds on how many test cases should be used. This study focus on the adequacy of penetration test case inputs for the SQL injection vulnerability. It builds an experimental platform to verify the proposed test case generation methods.

Reference [12] proposes and evaluates an AI-based pentesting system which makes use of RL to learn and reproduce average and complex pentesting activities. The scope is limited to network infrastructures PT planning and not the entire practice. Moreover, the authors tackle the complex problem of expertise capturing by allowing the learning module to store and reuse PT policies in a more efficient way.

The authors in [13] measured the performance of multiple fixed-strategy and learning-based agents. They concluded that Q-learning, with some extra techniques applied and greedy agent initialisation, performed best, surpassing human performance in the given environment.

Reference [14] suggests an automated penetration testing framework, based on deep learning techniques, particularly deep Q-learning networks (DQN). 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.

Reference [15] simplified the dynamics of SQLi vulnerabilities by casting the problem as security capture-the-flag and implementing it as an RL problem. Assuming that the vulnerability has been identified, they rely on RL algorithms to automate the process of exploiting SQLi. They implemented the model using two simulations, first simulation showed that a simple RL agent based on a Q-learning algorithm can successfully develop an effective strategy to solve the SQLi problem. A tabular Q-learning algorithm can discover a meaningful strategy by pure trial and error and can reach a performance close to the theoretical optimum. Using a table to store the Q-value function allowed them to carry out a close analysis of the learning dynamics of the agent, but this approach had poor

scalability. Thus, in the second simulation, they sacrificed interpretability in order to work around the issue of scalability. They deployed a deep Q-learning agent to tackle the same problem as in the first simulation. The deep Q-learning agents were able to learn a good strategy for the SQLi problem as well as provide a solution to the space constraints imposed by the instantiation of an explicit Q-table.

Given the success of RL in tackling and solving games, Penetration Testing, when distilled as a capture-the-flag (CTF), can be expressed as a game. However, in the case of penetration testing, an artificial agent may learn only by trial and error while a human hacker may rely on alternative sources of knowledge, deductions, 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 turn out to be a computationally hard challenge. Thus according to reference [16] injecting some form of elementary apriori knowledge about the structure of the problem may simplify the learning problem. Some basic forms of apriori knowledge which make the RL agent more efficient are lazy loading, state aggregation and imitation learning. 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. Simulation 1 solved the Port Scanning CTF problem using the basic tabular Q-learning algorithm. Solving this challenge required learning the problem meaning that the RL agent has to rely strongly on exploration. Simulation 2 solved the Non-stationary Port Scanning CTF problem by extending the previous problem by considering a more challenging scenario in which the target system is not stationary, but it may randomly change in response to the actions of the agent. Introducing non-stationary dynamics made the problem more challenging by preventing the agent from learning the exact structure of the problem with certainty. Despite this, the Q-learning agent was still able to solve the CTF problem in a reasonable yet sub-optimal way. Simulation 3 solved the Server Hacking CTF problem with Lazy Loading which considers a more realistic scenario. The problem presented a serious challenge to the tabular Q-learning agent because of the size of its Q-table. Relying on a priori knowledge in the form of lazy loading controlled the dimensionality of the state and action state pruning the non-relevant states. This method allowed the agent to

discriminate between relevant and non-relevant states based on its experience. Simulation 4 solved the Website Hacking CTF problem with State Aggregation preserving most of the complexity of Simulation 3. State aggregation allowed them to inject useful prior information about the structure of the problem, thus simplifying exploration and reducing the number of (state, action) pairs. Simulation 5 solved the Web Hacking CTF problem with Imitation Learning which emulates learning in a teacher-and-student setting, where expert paradigmatic behaviours are offered to a student to speed up its learning. Imitation learning proved to be an effective technique to enable faster learning for the RL agent. The improvement was due to the possibility of introducing the agent's knowledge of the structure of the problem. Instead of encoding knowledge of the structure of the problem in a formal mathematical way, they provided the RL agent with concrete observations about the structure of the problem. The agent could successfully exploit this information in order to learn an optimal policy.

Reference [17] considered the problem of exploiting SQLi vulnerabilities, representing it as a capture-the-flag scenario in which an attacker can submit strings to an input form with the aim of obtaining a flag token representing private information. The attacker was modelled as an RL agent that interacts with the server to learn an optimal policy leading to an exploit. 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 and 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.

REFERENCES:

[1] A taxonomy of SQL Injection Attack - https://sci-hub.se/10.1109/ICICM.2013.53

[2] SQL Injection: Classification and Prevention
https://sci-hub.se/10.1109/iciem51511.2021.9445347

[3] Efficient Solution for SQL Injection Attack Detection and Prevention:
https://www.ijsce.org/wp-content/uploads/papers/v3i1/A1399033113.pdf

[4] SQL Injection Prevention by Adaptive Algorithm
https://www.iosrjournals.org/iosr-jce/papers/Vol17-issue1/Version-3/E017131924.pdf (How SQL injections are done)

[5] SQL Injection Attack Prevention Based on Decision Tree Classification
-https://sci-hub.se/10.1109/ISCO.2015.7282227

[6] SQL Injection Attacks Predictive Analytics Using Supervised Machine Learning Techniques-https://ijcat.com/archieve/volume9/issue4/ijcatr09041004.pdf
[7] Detection of SQL injection based on artificial neural network
https://sci-hub.se/http://dx.doi.org/10.1016/j.knosys.2020.105528
[8] Machine Learning for SQL Injection Prevention on Server-Side Scripting https://sci-hub.se/10.1109/icsec.2016.7859950 (compared multiple algos)
[9] SQL Injection Detection Using Machine Learning Techniques and Multiple Data Sources-https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1649&context=etd_projects (compared multiple algos but more than above)
[10] Reinforcement Learning Model,Algorithms and Its Application https://sci-hub.se/10.1109/MEC.2011.6025669
[11] Attack model based penetration test for SQL injection vulnerability https://sci-hub.se/10.1109/COMPSACW.2012.108 (Penetration testing … writing effective sql injections)
[12] Reinforcement Learning for Efficient Network Penetration Testing-https://pdfs.semanticscholar.org/ca87/06c32cfa334d25f1fa7f521dc983636f81d6.pdf?_ga=2.126813839.306065165.1658468917-1177402474.1658468917
[13] Automating Penetration Testing using Reinforcement Learning https://stefann.eu/files/Automating%20Penetration%20Testing%20using%20Reinforcement%20Learning.pdf
[14] Automated Penetration Testing Using Deep Reinforcement Learning https://sci-hub.se/10.1109/EuroSPW51379.2020.00010
[15] Simulating SQL Injection Vulnerability Exploitation Using Q-Learning Reinforcement Learning Agents - https://arxiv.org/pdf/2101.03118.pdf
[16] Modelling Penetration Testing with Reinforcement Learning Using Capture-the-Flag Challenges-https://arxiv.org/pdf/2005.12632.pdf
[17] **https://link.springer.com/chapter/10.1007/978-3-030-91625-1_6 (SQL Injections and Reinforcement Learning: An Empirical Evaluation of the Role of Action Structure)**