An artificial intelligence model for the exploitation of SQL injection vulnerabilities in web applications

by Daoud Ahmed

B.S. in Information Technology (Cybersecurity), December 2019, Strayer University
M.S. in Digital Forensics, May 2023, University of The Cumberlands

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

August 15, 2025

Praxis directed by

Ould Mohamed Vadel
Professorial Lecturer of Engineering and Applied Science

The School of Engineering and Applied Science of The George Washington University certifies that Daoud Ahmed has passed the Final Examination for the degree of Doctor of Engineering as of July 10, 2025. This is the final and approved form of the Praxis.

An artificial intelligence model for the exploitation of SQL injection vulnerabilities in web applications

Daoud Ahmed

Praxis Research Committee:

Ould Mohamed Vadel, Professorial Lecturer in Engineering and Applied Science, Praxis Director

John M. Fossaceca, Professorial Lecturer in Engineering and Applied Science, Praxis Chair

Arthiraiyan Appulingam, Professorial Lecturer in Engineering and Applied Science, Committee Member

© Copyright 2025 by Daoud Ahmed
All rights reserved

Dedication

This research is dedicated to my parents and family. Without their unwavering support, it would have been impossible. The continuous prayers of my mother are the reason for this success. The life lessons taught by my father, Hafiz Ghulam Ahmed, especially how to work under pressure and stretch my limits, were vital during this journey. My wife, Sara, supported and encouraged me to accomplish my dream of a doctorate. My beloved 9-year-old daughter, Fatima, asked me every day how far I was completing my studies. My son, Omar, who is inspired by robots and wants to become a scientist, is fascinated by the idea of training robots. My newborn, Salahuddin, motivated me and made me smile, and my stress would fade just by looking at his beautiful smile.

Acknowledgements

I am deeply grateful to Allah Almighty for providing strength and resilience during the journey and making it both manageable and fulfilling.

Thanks to my advisor, Dr Ould Mohamed Vadel, for his valuable guidance and support throughout the course of this research.

Abstract of Praxis

The purpose of the research is to evaluate the existing Structured Query Language (SQL) injection (SQLi) exploitation tools and models based on an evaluation framework consisting of five dimensions of novelty, diversity, validity, semantic correctness, and Web Application Firewall (WAF) evasive capabilities. Then, sequence-based and feedback-directed AI models were used to generate payloads for in-depth analysis of their real-world applications and limitations. The purpose of this research is to discover unseen SQLi vulnerabilities in web applications by generating and applying new attack payloads. Therefore, research adopted an exploratory, experimental, and analytical approach. First, the limitations of existing SQLi exploitation solutions were identified through the exploration of the literature review and by applying tools against the simulated test environment. Then, Text Generative Adversarial Network (TextGAN) and stacked Long Short Term Memory (LSTM) standalone models were trained on the SQLi payloads dataset to generate novel, diverse, valid, and semantically correct payloads. The payloads generated were analyzed based on the proposed evaluation framework to test their effectiveness and real-world applicability. The results of the research showed that existing SQLi exploitation tools and models lack a balance between novelty, diversity, validity, semantic correctness, and WAF bypass. Stand-alone models without feedback can enhance the novelty and diversity of the generated payloads, but the syntax and effectiveness of the generated payloads were compromised. While feedback-based Artificial Intelligence (AI) models can provide a balance between these dimensions by enforcing novelty, diversity, validity, semantic correctness, and evasive capabilities in generated payloads. This directed

approach can yield unseen attack payloads to discover vulnerabilities in web applications and firewalls.

Table of Contents

Dedication iv

Acknowledgements v

Abstract of Praxis vi

List of Figures xi

List of Tables xiii

List of Symbols xiv

List of Acronyms xv

Chapter 1—Introduction 1

1.1 Background 1

1.2 Research Motivation 2

1.4 Problem Statement 3

1.4 Thesis Statement 4

1.5 Research Objectives 4

1.6 Research Questions and Hypotheses 5

1.7 Scope of Research 6

1.8 Research Limitations 7

1.9 Organization of Praxis 7

Chapter 2—Literature Review 8

2.1 Introduction 8

2.2 Application Penetration Testing and Its Role in SQLi Exploitation 8

2.2.1 Limitations of Manual Application Penetration Testing 9

2.2.2 Criteria for Evaluating Existing Tools and Approaches 11

2.3 Existing Solutions and Methods in Exploiting SQLi Vulnerabilities 11
2.4 Machine Learning Techniques in SQLi Exploitation 17
2.5 Summary and Conclusion 27
Chapter 3—Methodology 29
3.1 Introduction 29
3.1.1 Overview of Methodology 29
3.2 Data Collection 30
3.2.1 Dataset and Preprocessing 30
3.3 Model Design 31
3.3.1 - TextGAN Architecture for SQLi Attack Payloads 31
3.3.2 Adversarial Training Loop for GANs 33
3.3.3 Generating Novel SQLi Payloads 34
3.4 – Stacked LSTM- A Sequence-Based Generator for SQLi Payloads 35
3.4.1 Model Architecture 35
3.4.2 Model Training 37
3.4.3 SQLi Payload Generation 38
3. 5 Evaluation Metric Embedded RL-LSTM Model 38
3.6. Experimental Setup 40
3.5.1 - Deployment 40
3.5.2 - Testing 41
3.7 - Evaluation Metric 42
3.7 Limitations 43
3.8 Conclusion 44

Chapter 4—Results 46
4.1 Introduction 46
4.2 TextGAN and Stacked LSTM Model Training Performance 47
4.2.1 Training Behavior of TextGAN and Stacked LSTM 48
4.2.1 Evaluation of Generated Payloads Based on Defined Metric 50
4.3 Comparative Summary of Models with Benchmark Tools 57
4.4 Overcoming the Issue of Validity with an Evaluation Metric Feedback 59
4.4.1 Training Graph of Model 60
4.5 Comparison of RL LSTM with Text GAN, Stacked LSTM, and SQLmap ... 71
Chapter 5—Discussion and Conclusions 74
5.1 Discussion 74
5.2 Conclusions 76
5.3 Contributions to Body of Knowledge 77
5.4 Recommendations for Future Research 78
References 79

List of Figures

Figure 1. DeepSQLi Workflow, adopted from Liu et al., (2020). 20

Figure 2. Vanilla GANs architecture, sourced from Dunmore et al., (2023) 23

Figure 3. GANs and GA based SQLi Attack Sample Generation Method, derived from Lu et al. (2022)..... 24

Figure 4. Reinforcement Learning based Framework for XSS Attack Generation, based on Chowdhary et al. (2023) 26

Figure 5. High-Level Overview of Research Methodology 30

Figure 6. GAN architecture consists of a generator and a discriminator..... 31

Figure 7. LSTM architecture for SQLi Payload 35

Figure 8. Evaluation Embedded RL - LSTM architecture for SQLi Payloads..... 39

Figure 9. Environment Setup - Deployment and Testing Phase of Methodology 40

Figure 10. Training Curves of TextGAN and Stacked LSTM..... 48

Figure 11. Novelty of Generated Payloads 51

Figure 12. Diversity Metrics 52

Figure 13. Existing AI solutions evaluated based on Evaluation Framework..... 57

Figure 14. AI models developed to address the limitations of Existing AI solutions, and their performance was compared with the Benchmarking Tool..... 58

Figure 15. All dimensions at Single Seed Payload..... 61

Figure 16. All dimensions at the Batch of Seed Payloads 62

Figure 17. Reward graph at a Single Seed..... 62

Figure 18. Reward Progression at Batch Payload..... 63

Figure 19. Novelty at Single Payload 63

Figure 20. Novelty at Batch of Payloads 64
Figure 21. Diversity at a Single Seed Payload..... 65
Figure 22. Diversity at a Batch of Payloads 65
Figure 23. Validity of Generated Payloads at a Batch of Seed Payloads 66
Figure 24. Semantic of Generated Payloads at a Single Seed Payload 67
Figure 25. WAF Bypass at the Single Seed Payload..... 67
Figure 26. WAF Bypass at the Batch of Seed Payloads..... 68
Figure 27. WAF Bypass Over Time at a Batch Payload 68
Figure 28. Comparison of Bypassed and Blocked Payloads 70
Figure 29. Model Comparison Based on Evaluation Metric 73

List of Tables

Table 1. Penetration Testing Stages derived from PTES (2014) 9

Table 2. Generator Hyperparameters 32

Table 3. Hyperparameters for Training 34

Table 4. Model Optimization Parameters 34

Table 5. Network Architecture Parameters 37

Table 6. Evaluation Metric for SQLi Payload Generation 42

Table 7. Novelty comparison of stacked LSTM and TextGAN 51

Table 8. Summary of Diversity Measures Applied to Generated SQLi Payload 52

Table 9. Parser-Based Validity of Generated SQLi Payloads 53

Table 10. Payloads generated by TextGAN 54

Table 11. Stacked LSTM Generated Payloads 56

Table 12. WAF Bypass Comparison of Stacked LSTM and TextGAN 57

Table 13. Payloads generated by a single seed and a batch of payloads 66

Table 14. WAF Bypass Progress Over Time at a Single vs a Batch of Payloads 69

Table 15. Payloads Generated by Model to Evade WAF 69

Table 16. Comparison of SQLi Tools and Models 72

List of Symbols

t	Time-step index
T	Sequence length
z_t	Noise input at time step t
I	Identity matrix
$Emb(\cdot)$	Embedding function that maps input tokens to continuous vector space
h_t^G	Generator's GRU hidden state at time t
h_{t-1}^G	Generator's hidden state from previous time step
$GRU_{G(\cdot)}$	GRU unit used in the generator
W_G	Generator's output weight matrix
b_G	Generator's output bias vector
y_t	Token probability distribution at time t after softmax
$G(z)$	Full output sequence from the generator $\{\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_T\}$
$GRU_{D(\cdot)}$	GRU unit used in the discriminator
x_t	Real token from the dataset at time t
e_t	Embedded version of x_t for the discriminator
h_t^D	Discriminator's GRU hidden state at time t
h_{t-1}^D	Previous hidden state in the discriminator
W_D	Discriminator's output weight vector
b_D	Discriminator's output bias scalar
$D(x)$	Discriminator's output probability: real (1) or fake (0)
R	Overall reward signal used for reinforcement or evaluation
$\sigma(\cdot)$	Sigmoid activation function
α_i	Weight assigned to the $\dot{l}$th evaluation dimension

List of Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
CIA triad	Confidentiality, Integrity and Availability
CTF	Capture-the-Flag
DAST	Dynamic Application Security Testing
DCGAN	Deep Convolutional Generative Adversarial Network
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
GA	Genetic Algorithms
ICS	Industrial Control Systems
IDS	Intrusion Detection Systems
LSTM	Long Short-Term Memory
LSTM-ED	Long Short-Term Memory-based Encoder-Decoder
LSTM-EVI	Long Short-Term Memory Recurrent Neural Network-Enabled Vulnerability Identification
ML	Machine Learning
NLP	Natural Language Processing
OWASP	Open Worldwide Application Security Project
PTES	Penetration Testing Execution Standard
RNN	Recurrent Neural Network
RL	Reinforcement Learning
SAST	Static Application Security Testing

SQL	Structured Query Language
TPU	Tensor Processing Unit
WAF	Web Application Firewall
XSS	Cross-Site Scripting
ZAP	Zed Attack Proxy

Chapter 1—Introduction

1.1 Background

Nowadays, web applications are connected to backend databases to store customer information and data. This integration of applications and databases allows users to conveniently place orders, shop online, and pay bills, which are a subset of activities and tasks within a wider context. They are the backbone of modern e-commerce and banking web applications and are almost a part of any industry. However, this type of smart integration between applications and databases can open the door for threats and vulnerabilities. One of the most notorious security flaws that affect database-driven applications is the Structured Query Language (SQL) injection (SQLi) attack. SQLi vulnerabilities occur in the database when the user input is not validated properly. The improperly validated input is directly inserted into a database query (Nguyen-Duc et al., 2021). The backend database executes maliciously inserted code without proper validation, which gives access to the attacker to bypass authentication mechanisms and access sensitive data (Hajar et al., 2024).

SQLi is still a security threat in database-driven web applications, and they represent nearly a quarter (23.4%) of all the critical vulnerabilities in web applications (Borgeaud, 2024). Hackers leverage SQLi flaws for their own benefit and exploit them to exfiltrate data from back-end databases. SQL injection vulnerabilities negatively impact the US economy and cause $10 billion in losses every year (Zhang et al., 2022). In addition, SQLi vulnerabilities can cause the CIA triad to be completely compromised. For instance, the disclosure of sensitive information from the database without authorization can disclose sensitive information and data. The unauthorized access can further grant full access to

hackers to modify or delete data residing on the database, which is a complete destruction of integrity. While the time-based SQLi attacks can allow attackers to disrupt the services for a certain time by delaying the database response, and as a result, availability will be impacted. Therefore, the threats and impacts on the database and application security demand robust security measures to protect against SQL injection attacks (OWASP, 2024).

SQLi vulnerabilities have impacted many businesses in recent times. For instance, malicious actors actively exploit SQL injection vulnerabilities in MOVEit, a widely utilized file transfer software, resulting in substantial data compromise across numerous organizations (Vijayan, 2023). Owing to this ongoing large-scale cyberattack, over 100,000 individuals have been notified that their personal information has been obtained by unauthorized parties (Tidy, 2023). In another SQLi attack, the GambleForce hacking group targeted over 20 websites in various sectors across eight countries and exploited CMS vulnerabilities to access sensitive information. They comprised 24 organizations, breaching six websites in Australia, Indonesia, the Philippines, and South Korea. Another hacker group named ResumeLooters compromised 65 websites and stole user databases containing personal and employment information of job seekers, with 2,188,444 rows of data, including 510,259 user entries (Rostovcev, 2024). These incidents justify that SQLi is still impacting the security of applications and can result in data breaches.

1.2 Research Motivation

The most recent SQLi data breaches and their negative impact on security are the motivation behind this research. Various questions arise from the recent SQLi breaches that are the driving force behind this work and directing research. Secondly, the motivation is to find an AI-based exploitation model that can discover unseen vulnerabilities. As a

result, the attack surface will be reduced. Therefore, the use of AI is essential because traditional exploitation processes can overlook unconventional attack vectors due to human imagination or experience limitations (Hilario et al., 2024). To execute successful penetration testing, existing tools must be enhanced by leveraging artificial intelligence models to render testing more efficient and effective (Chowdhary et al., 2023). So, integrating machine learning algorithms and artificial intelligence models into the penetration testing exploitation phase will significantly enhance the efficiency and effectiveness of web application security and, as a result, improve the target system's overall security posture of the system will improve (Hajar et al., 2024). AI has been introduced in many modern penetration testing tools and methods. For instance, a machine learning-based penetration testing tool named GoyiThon (Jagamogan et al., 2022) has demonstrated better results in detection and exploitation than traditional tools (Hajar et al., 2024). In addition, a penetration testing approach based on AI models proposed by Chowdhary et al. (2023) showed advantages in web application security by evading WAF without human intervention. In a nutshell, the integration of AI into the exploitation phase will discover complex and unseen SQLi vulnerabilities that humans might overlook.

1.4 Problem Statement

SQL Injection vulnerabilities are a constant risk to database-driven web applications (Nair, 2024), accounting for 23% of the most common critical vulnerabilities in internet-facing web applications (Borgeaud, 2024).

The most recent SQLi data breaches show that SQLi vulnerabilities can cause risks to the security and integrity of database-driven web applications. Organizations can suffer losses due to SQLi in the form of data breaches, financial losses, and reputational damage. SQLi

vulnerabilities are still part of OWASP's top 10 list under the injection attack category (OWASP, 2021). Mitigating SQL injection vulnerabilities requires a layered approach along with the industry's best practices, including input validation and parametrized queries. The addition of machine learning techniques to the approach will help discover undetected and hidden SQLi vulnerabilities in web applications.

1.4 Thesis Statement

An AI-based model will enhance the SQL Injection vulnerability exploitation rate by overcoming the limitations of existing solutions.

An AI-based model will enhance the SQL Injection vulnerability exploitation rate by overcoming the limitations of existing solutions. Because AI-based models will generate new SQLi attack vectors that can discover unseen attacks in database-driven applications. GyoiThon and autonomous GAN-based penetration testing approaches are practical examples of AI integration in offensive security. In addition, the AI can assist in automation, which will reduce the time of generating evasive and effective payloads. Therefore, the integration of AI can greatly assist in the automation of crafting novel SQLi attack payloads.

1.5 Research Objectives

The purpose of this research is to analyze performance of the existing SQLi payloads generation and exploitation models based on an evaluation framework and to develop models using same evaluation framework to enhance the novelty of generated SQL injection (SQLi) attack payloads using Text Generative Adversarial Networks (GANs) and a stacked Long Short-Term Memory (LSTM) model. In this way, the limitations of existing

tools and models will be identified, and then an exploitation model will be proposed for better performance that overcomes the weaknesses of existing solutions.

1.6 Research Questions and Hypotheses

RQ1: What are the limitations of existing SQLi exploitation tools in exploiting complex SQL Injection vulnerabilities?

RQ2: How can AI models improve the novelty, diversity, validity, and effectiveness of SQL injection attack payloads during exploitation while keeping a balance?

RQ3: Can reinforcement learning and stacked LSTM with an evaluation-feedback loop improve the validity and WAF evasion of ML-generated SQLi payloads?

RQ4: What is the impact of the input seed payloads on novelty, diversity, and WAF bypass performance in reward-based models?

H1: Existing SQLi exploitation tools are unable to generate novel, diverse, and WAF-evading attack payloads while maintaining a balanced approach between them.

H2: The sequence-based AI models such as TextGAN and stacked LSTM, can increase the novelty, diversity, and validity in SQLi attack payloads while maintaining a balance compared to those produced by existing SQLi solutions.

H3: A feedback reward function based on the evaluation metric through reinforcement learning in stacked LSTM, will enhance the WAF bypass rate.

H4: The use of multiple input seed payloads can introduce novelty and diversity in input data, and as a result, the WAF bypass rate will increase compared to a single seed payload.

The aim of the research is to study the limitations of existing solutions exploiting SQL injection (SQLi) vulnerabilities during penetration testing based on an evaluation framework. Then, it proposes machine-learning models to address the identified gaps in solutions, and the performance of generated payloads is analyzed using the same evaluation framework to preserve impartiality. Addressing these gaps will enhance the exploitation phase of SQLi by integrating machine learning techniques and algorithms. The study is focused on TextGAN and stacked LSTM machine learning techniques to address the gaps in the existing SQLi exploitation tools. The research further investigates how these algorithms improve the capabilities of existing solutions based on the defined metric. The successful integration of artificial intelligence in developing novel attack payloads during SQLi exploitation using machine learning is expected to improve the overall penetration testing process by reducing the time and effort of crafting novel and evasive payloads.

1.7 Scope of Research

The study focuses on the exploitation phase of SQL injection (SQLi) by generating novel, diverse, valid, semantically correct, and evasive payloads. As a result, the human intervention in crafting novel payloads will be reduced. This effort will support offensive security teams in penetration testing to exploit SQLi vulnerabilities more efficiently and effectively. Therefore, it is assumed that the target application is vulnerable to SQLi. The research narrows its scope to address the exploitation phase only, with the aim of developing methods and tools to automate and improve this critical step in penetration testing.

1.8 Research Limitations

The research was conducted using PHP and the backend MariaDB database. Due to this, the outcomes may differ for applications developed in alternative programming languages. This consideration can be extended to various types of databases. Furthermore, blind SQLi and out-of-band SQLi were excluded from this study. In addition, limited access to datasets containing vulnerable payloads may influence the performance of AI models.

1.9 Organization of Praxis

Praxis integrated Generative Adversarial Networks (GANs) and Long Short-Term Memory (LSTM) models to automate exploitation while generating novel attack payloads. Chapter 1 introduces the study, and Chapter 2 provides a comprehensive overview of the existing SQL injection (SQLi) exploitation methods and their limitations. Section 3 consists of the methodology section and explains the research process and configuration of the simulated environment. Section 4 presents the study's results, and Section 5 presents the conclusions and recommendations for future research.

Chapter 2—Literature Review

2.1 Introduction

This chapter provides a bird's-eye view of different penetration testing stages and their roles in application security. Subsequently, it compares manual and automated penetration testing and their limitations. After that, the criteria for this praxis research are developed and listed to evaluate the most relevant tools and studies. The following section focuses on a critical analysis of studies related to SQLi exploitation based on outlined guidelines, including existing and ML-based solutions and approaches. All studies are evaluated based on established benchmarks to identify the gaps and find an opportunity for this research. Lastly, the conclusion part of this chapter summarizes the shortfalls in the assessed studies and addresses them in the following chapters.

2.2 Application Penetration Testing and Its Role in SQLi Exploitation.

Penetration testing is a simulated cyber-attack on web applications to identify vulnerabilities in applications before an attacker could exploit them (Hajar et al, 2024). The purpose of conducting such tests is to evaluate and enhance the effectiveness of the security posture of an organization's applications (Stefinko et al., 2016) and is considered one of the most effective approaches for uncovering SQLi vulnerabilities in web applications (Anaoval et al., 2024). Penetration testing is carried out through two approaches: manually and automatically using automated tools. The manual approach is executed by a human security tester and is regarded as one of the most effective and prevalent techniques due to human involvement in identifying vulnerabilities (Stefinko et al., 2016). Automated testing is performed by tools to speed up the process of testing; however, these tools may generate false positives (Rane & Qureshi, 2024). Web application penetration testing is conducted

in phases through a systematic approach. The Penetration Testing Execution Standard (PTES) outlines penetration testing into seven distinct phases (PTES, 2014) as shown in Table 1.

Table 1. Penetration Testing Stages derived from PTES (2014)

Phase	Name
1	Pre-engagement Interactions
2	Intelligence Gathering
3	Threat Modeling
4	Vulnerability Analysis
5	Exploitation
6	Post Exploitation
7	Reporting

This research was solely focused on the exploitation phase, and all other phases were outside the scope of this study. The primary objective of exploitation is to gain unauthorized access to a system or resource by circumventing established security measures. The countermeasures for the exploitation phase include Web Application Firewalls (WAF) to mitigate attacks against applications and databases (PTEST, 2014). Each phase is executed through the utilization of tools to reduce time and manual effort.

2.2.1 Limitations of Manual Application Penetration Testing

The penetration testing process is time-consuming and highly dependent upon the skill set of the tester. Each testing engagement has a defined time period allocated at the beginning of testing. Attackers have more time than penetration testers, and they may spend months or even years planning and organizing their attacks, while penetration testers

operate within a constrained time frame to deliver their test findings (Mamilla, 2021). Testers have to perform meticulous, complex, and time-consuming tasks during comprehensive penetration testing (Stefinko et al., 2016). Therefore, creating novel attack payloads during testing consumes a significant amount of valuable time for testers.

In addition to that, the quality of the penetration test is dependent on the skill set of the tester (Mamilla, 2021). If the tester lacks experience in performing testing, there is a high chance that vulnerabilities may remain hidden and undetected. Attackers can exploit these undetected vulnerabilities to gain unauthorized access to systems and applications. Therefore, an approach is required to overcome the challenge of hidden issues to generate novel payloads that are unseen, evasive, and quickly available for effective testing. This objective can be achieved through the application of AI models. As a result of AI automation, the time spent on generating payloads will be reduced, and the expertise required from testers to develop exploits and payloads to effectively identify potential unseen vulnerabilities will be reduced. As a result, the quality of the work will improve.

Taking that into consideration, the objective became narrowed to enhance the exploitation phase for SQLi attacks in two ways: The proposed approach should generate novel attack payloads that have the ability to evade defensive mechanisms for testing purposes using machine learning algorithms in minimal time. Secondly, it should reduce the manual work for penetration testers through automation while maintaining the quality of the work. Furthermore, it is to ensure that the proposed methodology is practical and applicable in real-world scenarios. Consequently, evaluation criteria were developed specifically for this research to find limitations and gaps in tools and studies to formulate an improved version of the approach using ML algorithms that has the potential to reduce

time and effort. These specifications have led to the development of evaluation criteria that were developed specifically for this research to find limitations and gaps in tools and studies to formulate an improved version of the approach using ML algorithms that has the potential to reduce time and effort.

2.2.2 Criteria for Evaluating Existing Tools and Approaches

A criterion was established specifically for this research to evaluate existing research papers, studies, and tools to find limitations of techniques and approaches. The following criterion directed the development of an enhanced version of the approach to address these limitations.

A. Does the tool/model generate novel attack payloads for SQLi exploitation?

B. Do existing tools/models have the ability to mutate SQLi payloads for different variations that can be used to test firewalls (WAF)?

C. Do these tools/models have the potential to reduce the time and number of steps required for manual SQLi exploitation?

D. Are these tools/models applicable in real-world environments?

E. Can these models be integrated into existing solutions?

2.3 Existing Solutions and Methods in Exploiting SQLi Vulnerabilities

Traditional and most common techniques to detect SQLi vulnerabilities in applications are static analysis, dynamic analysis, fuzzing (Harb et al., 2021; Zhao, 2020), and penetration testing. Static Application Security Testing (SAST) is a quality assurance technique that ensures code quality and identifies SQL vulnerabilities. In this approach, the

source code is analyzed to identify potential issues by examining it without execution by checking the code structure and logic flows of statements. For this purpose, various SAST tools work as white-box testing tools without running the source code. The basic techniques used in SAST tools include (1) synthetic analysis, such as calling insecure API functions or using insecure configurations. In contrast, semantic analysis requires an understanding of program semantics, that is, data flow or control flow. SATS tools use lexical analysis as a more comprehensive and complex analysis technique to identify SQLi. This technique is useful to find vulnerabilities, but it generates false positives (Elder, 2022), making it cumbersome for security teams to validate each issue individually. Therefore, the advised approach is to combine multiple tools for the best performance during SAST assessments (Nguyen-Duc et al., 2021). However, this adds extra cost to the application security.

On the other hand, Dynamic Application Security Testing (DAST) uses automated tools for dynamic analysis to detect SQLi vulnerabilities. DAST tools cannot access the source code and generate a set of malformed inputs by applying security rules. OWASP Zed Attack Proxy (OWASP ZAP) and Burp Suite are used in DAST testing in industry settings. The main drawback of this technique is that it leaves undetected vulnerabilities in the application (Elder et al., 2022), which can open the door for adversaries to gain unauthorized access to the data. Fuzzing is a technique that involves providing invalid, unexpected, or random data to the input fields of an application to identify bugs and vulnerabilities. It is efficient, highly automated, and scalable, making it a widely utilized method for discovering vulnerabilities in web applications and other systems. Fuzzing can be conducted manually and automatically using specialized tools. Manual fuzzing is a time-consuming process; therefore, Burp Suite and SQLmap are utilized to automate SQLi

detection through fuzzing (Zhao, 2020). This gap is filled with performing penetration testing, which is typically a human-driven procedure that requires deep knowledge of possible attacks and hacking tools that can be used to launch the tests (Casola et al., 2020). The goal is to identify and patch security holes before an attacker discovers them (Chowdhary et al., 2023). The methodology for penetration testing consists of planning and scoping, Reconnaissance, Vulnerability analysis, Exploitation, Post-exploitation, and Reporting (Hajar et al., 2024). Application security requires penetration testing (Alanda et al., 2021) to reduce the attack surface.

Penetration testing is the most effective method in identifying vulnerabilities in applications because it discloses hidden issues (Elder, 2022), but it is less efficient due to manual work compared to automated techniques such as SAST and DAST. Therefore, a comprehensive security assessment strategy is the combination of automated and manual techniques that combine all of them for effectiveness and efficiency.

Several methods and approaches have been proposed for detecting SQL injection vulnerabilities by using a combination of tools that predominantly rely on the principles of fuzzing. For instance, Ojagbule et al. (2018) proposed a methodology to identify and exploit SQLi utilizing Nikto and SQLmap. An initial scan was conducted employing Nikto, followed by the utilization of SQLmap in the exploitation phase of penetration testing to retrieve sensitive information from the database. SQLmap was employed to execute commands to extract database details, tables, and column structures, as well as to retrieve sensitive data such as login credentials (Abdullah et al, 2023). The combination of both tools automated the process of vulnerability analysis and exploitation phases to reduce time and steps; however, the effectiveness of SQLmap depends on manually created rules,

which limit its adaptability to novel attack patterns (Liu et al., 2020). While Nikto lacks the advanced features required for extensive deployments, it is more suitable for basic vulnerability assessments (Baskaran et al., 2024). In short, the proposed approach used tools that are unable to generate novel attack payloads to test applications against sophisticated attacks based on the evaluation criteria.

Rose et al. (2022) used a combination of several scanning tools for the purpose of improving and enhancing the crawling capabilities of existing tools. This approach leveraged automated scanning tools to navigate and analyze applications comprehensively for common injection attacks. This enhancement addressed the limitations of traditional scanners' crawling capabilities and increased vulnerability detection. This research improves the vulnerability analysis phase of penetration testing by identifying all injection points; however, it does not address the exploitation phase. Furthermore, the time and number of steps are not reduced in the exploitation phase. Also, the scanner was unable to generate novel attack vectors to exploit injection vulnerabilities. Not only that, but the proposed scanner also applied fuzzing, which suffers from issues of limited diversity in test cases. In traditional fuzz testing, test cases are manually written or randomly generated, which can be inadequate to meet the complex and dynamic requirements of contemporary web fuzz testing (Lin & Dong, 2024).

Bouafia et al.'s (2023) methodology relied on a combination of Accunetix, Burp Suite, and SQLmap for the identification and exploitation of SQL injection vulnerabilities to cover all stages of testing. Initially, Acunetix, which is known for its fast performance and detailed assessments (Baskaran et al., 2024), was utilized to perform an automated scan to effectively identify SQLi vulnerabilities (Rahman et al., 2020) in web applications. Burp

Suite, an extensive tool specially designed and applied for conducting application security assessments (PortSwigger, 2024), was used to inspect and modify HTTP requests and traffic between the browser and the application. This tool has shown high performance in crawling web applications (Althunayyan et al., 2022) in various studies. When the inspection and modification phase was completed, Burp Suite's CO2 extension was used to automatically generate SQL commands for creating SQL payloads. These SQL commands were then configured with SQLmap to exploit the identified vulnerabilities by Accunetix. The integration of multiple well-known tools automates various stages of application testing, making it a powerful approach that can be scaled up and implemented in real-world environments. Consequently, the duration of testing and the number of steps were reduced to a certain extent. However, the exploitation phase was conducted by SQLmap in this approach, and this tool suffers from inherent limitations. First of all, it is based on fuzzing, and fuzzers rely on dictionaries rather than generating random inputs or mutated data (Zhao, 2020). Secondly, SQLmap is unable to generate novel attack payloads due to its dependence on predefined syntax. This limitation of SQLmap prevents it learning and discovering new SQL injection attacks. In other words, the effectiveness of the tool is entirely contingent on manually created rules. That is why this limitation restricts its adaptability to novel attack patterns (Liu et al., 2020). While Acunetix may not be financially viable for smaller organizations (Baskaran et al., 2024).

Hajar et al. (2024) conducted a recent study on practical SQLi attacks utilizing a systematic approach to manual and automated penetration testing methods. Researchers thoroughly covered all stages of penetration testing, including planning, reconnaissance, vulnerability analysis, exploitation, post-exploitation, and reporting. The exploitation

phase of testing was carried out using SQLmap to automate injection testing and payload generation. This approach was practical and applicable in real-world environments due to the combined manual and automated testing methods. However, the exploitation process was completely based on SQLmap, which shows limitations in generating novel attack payloads. Consequently, the proposed approach did not fully satisfy the evaluation criteria for creating novel attack payloads.

Anaoval et al. (2024) took a manual approach to exploit SQL injection (SQLi) vulnerabilities in web applications. The 'OR'1'='1 technique was used to bypass authentication by injecting SQL code into input fields. The manual and step-by-step testing methodologies used and applied in this approach were effective than automated scripts and vulnerability scanners in identifying security vulnerabilities (Singh et al., 2020). But creating and developing payloads in a manual way is time-consuming due to manual efforts, and it is heavily dependent on the experience and expertise of the tester (Rane & Qureshi, 2024). Also, this technique only works if input validation is missing, which restricts the effectiveness of this method in identifying complex and hidden vulnerabilities.

The tools and techniques discussed and explained so far include SQLmap, Burp Suite, Acunetix, Nikto, and a systematic manual approach. These techniques are the most commonly utilized methodologies for identifying and exploiting SQLi vulnerabilities. The tools and techniques integrated in these methodologies enhance the identification and exploitation phases by reducing the time required for manually crafting injection payloads. However, they rely on predefined dictionaries rather than generating random inputs or altering existing data (Zhao, 2020). This reliance limits their capability to create novel attack payloads, thereby impeding their ability to learn or actively identify new SQL

injection attacks. Their effectiveness is constrained by manually crafted rules, making it harder for them to detect emerging attack patterns (Liu et al., 2020). Furthermore, fuzzy logic demonstrates poor scalability as the complexity and size of input data increase (Chowdhary et al, 2023), resulting in a significant decrease in the efficiency of systems employing fuzzy logic. Additionally, the aforementioned tools and techniques lack the capability to mutate specific attack payloads using machine learning algorithms to evaluate the security of defensive mechanisms such as WAF.

2.4 Machine Learning Techniques in SQLi Exploitation

The limitations of existing tools and methodologies motivated and encouraged researchers to design and develop advanced tools for generating novel and unseen attack payloads through the application of machine learning. Various approaches have been proposed to address and overcome the challenge of unseen attack vectors in exploiting SQLi vulnerabilities by using machine learning techniques.

Long Short-Term Memory (LSTM) and Generative Adversarial Networks (GANs) were employed in penetration testing for the identification and exploitation of SQLi attacks. LSTMs are a type of recurrent neural network (RNN) designed to address the exploding and vanishing gradient problems that occur when learning long-term dependencies in sequential data (Van Houdt et al., 2020). Furthermore, LSTMs show resilience against noisy data in security-related tasks, making them a suitable choice for generating effective and complex payloads for testing security vulnerabilities. LSTMs can also be utilized to create attack samples similar to GANs to enhance the performance of detection models.

Koroniotis et al. (2021) presented a deep learning-enabled penetration testing framework named Long Short-Term Memory Recurrent Neural Network-Enabled Vulnerability Identification (LSTM-EVI). Joshi et al. (2022) proposed employing Long Short-Term Memory (LSTM) neural networks to identify and prevent SQLi attacks. The results indicate a significant improvement in detection accuracy compared to blacklist filtering, regular expression, manual testing, and scanning. Liu and Dai (2024) adopted a hybrid approach to improve the detection of sophisticated and advanced SQL injection attacks. Thaqi et al. (2022) investigated machine learning techniques to enhance Burp Suite SQLi detection capabilities using Logistic Regression, Random Forest, Gradient Boosting, and LSTM machine learning models. Out of all algorithms, the LSTM algorithm provided the highest accuracy for SQL injection (SQLi) detection. Liu et al. (2024) proposed a novel approach to adversarial sample attacks and defenses in industrial control systems (ICS) utilizing a Long Short-Term Memory-based Encoder-Decoder (LSTM-ED). The LSTM-ED model effectively created realistic and protocol-conforming adversarial samples with high efficiency. These novel attack vectors demonstrated the capacity to reduce the performance of detection models through innovative attacks. This approach addressed the challenges of generating protocol-specific adversarial samples. This method was useful in designing defense models without prior information about these adversarial samples. These studies support that LSTMs have the potential to enhance the efficiency of the exploitation phase during penetration testing by reducing time and the number of steps. Furthermore, these findings highlight the practical utility of the proposed approach in strengthening system security by generating realistic and novel attack payloads.

Therefore, it is concluded from these studies that LSTMs were mainly used in the reconnaissance and identification stages of penetration testing for the purpose of detecting SQLi attacks. The LSTMs are suitable for SQLi attack payloads due to their ability to retain sequential patterns. The models with sequence learning capabilities are necessary for analyzing structured query language to enhance and improve database security (Joshi et al., 2022). Also, LSTMs can generate adversarial attack payloads in IoT systems (Liu et al., 2024) that have the potential to detect zero-day vulnerabilities in the environment. Therefore, they will be suitable for novel attacks due to their capability to learn long-term dependencies and sequential patterns in input data. Additionally, these studies provide a solid foundation that can serve as a reference for implementing LSTMs to improve the quality of generated novel attack payloads for the exploitation phase, thereby enhancing the efficiency and effectiveness of testing.

(Liu et al., 2020) proposed and developed a deep natural language processing (NLP) based tool to address the limitations and drawbacks of SQLmap. The architecture of the tool is shown in Figure 1. A transformer architecture neural language model was integrated into the tool to facilitate the discovery of both common and sophisticated SQLi variants during testing.

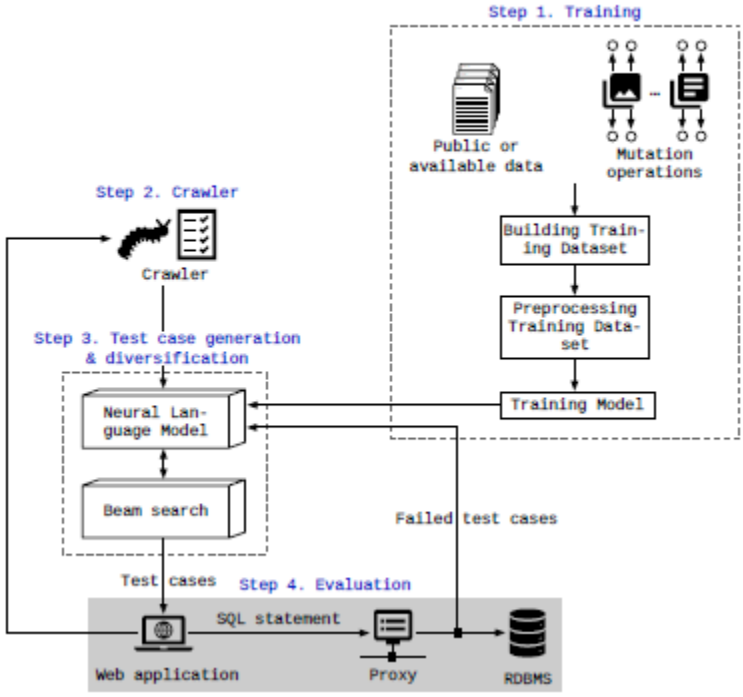


Figure 1. DeepSQLi Workflow, adopted from Liu et al., (2020).

The proposed tool was effective and efficient and showed improvements over SQLmap. First, it operated significantly faster than SQLmap and achieved better performance of up to six times. DeepSQLi employed fewer test cases, ensuring more rapid execution and making it a more resource-efficient tool. Also, it uncovers deeply hidden and previously unidentified vulnerabilities in the applications. Under advanced input validation scenarios, it consistently delivers superior results compared to SQLmap, which frequently fails to detect vulnerabilities in numerous cases. That is how it surpassed SQLmap in its detection capabilities to uncover complex vulnerabilities.

Secondly, it identifies and exploits a higher number of SQL injection (SQLi) vulnerabilities while utilizing testing resources more effectively and achieves higher exploitation rates. Transformer architecture enabled DeepSQLi to generate diverse,

semantically meaningful test cases to support the detection of non-traditional attack patterns that traditional tools may overlook and produce multiple semantically related test cases for each input. The innovation introduced in the tool for generating novel test cases addresses SQLmap's limitation of fixed and less adaptable test case generation. In addition to that, the tool possesses the capability to transform existing test cases into semantically related yet more sophisticated SQLi attacks. The iterative and cyclic feedback mechanism refines test cases to reveal previously unknown and deeply hidden vulnerabilities.

However, the integration of DeepSQLi with existing tools requires verification. The integration with existing tools is essential to leverage the capabilities of tools to identify other types of vulnerabilities and streamline the testing process in identifying and exploiting other types of vulnerabilities. Also, it has not been tested against Web Application Firewalls (WAFs) in real-world environments. In such enterprise environments, websites frequently deploy WAFs and Intrusion Detection Systems (IDSs) to counter attacks against applications.

Erdődi et al. (2021) adopted and used a reinforcement learning approach to explore SQL injection vulnerabilities. This approach introduced novelty to some extent, and validity was judged by whether the agent triggered the expected system response. However, they didn’t focus on semantic correctness or WAF evasion. Sommervoll (2023) proposed a similar reinforcement learning method to exploit SQL injection vulnerabilities in web applications. The methodology was based on and founded on modeling SQL injection (SQLi) vulnerability exploitation as a reinforcement learning (RL) problem. The researchers conceptualized SQLi as a Capture-the-Flag (CTF) challenge to retrieve a flag stored in a database by exploiting vulnerabilities in SQL queries. The methodologies

demonstrated potential for penetration testing, but further development is required for addressing complex, real-world SQL injection scenarios.

Even though these RL-based methodologies demonstrated and showed potential for penetration testing, further development is still required for addressing complex, real-world SQL injection scenarios. RL-based agents frequently explore their environment by taking numerous exploration actions, and this type of exploration leads to inefficiencies when compared to pre-defined decision tree approaches. This results in longer penetration testing sessions. A higher number of actions against the test environment increases the time but makes the testing detectable by intrusion detection systems. This type of intrusive and intensive exploration, due to the number of actions during testing, reduces and diminishes the stealthiness of the penetration testing engagements. Since the RL agents are trained in controlled, simulated, and simplified scenarios, and when applied to real-world networks with complex topologies and unexpected behaviors, the agent might need to take additional and unnecessary actions to adapt (Becker et al., 2024). Lastly, the proposed approach cannot be integrated into the existing application penetration testing frameworks and tools except for its specific use of SQLi exploitation.

To circumvent defensive measures and WAFs, obfuscation and mutation of statements are essential to enhance the efficacy of the payloads (Lu et al., 2022). Generative Adversarial Networks (GANs) produce adversarial samples by constructing feature vectors that deceive malware classifiers into categorizing malicious programs as benign, thereby facilitating evasion (Renjith et al., 2021). GANs play the role of evaluators by generating adversarial attack payloads to evade detection and test the machine learning classifiers effectively (Rafiq et al., 2022). The capability of GANs to yield high-quality results across

diverse applications demonstrates their viability as a framework for generative modeling (Goodfellow et al., 2014). The architecture of Vanilla GAN is shown in Figure 2.

Generative Adversarial Networks (GANs) constitute an innovative approach to estimating generative models through an adversarial learning framework. The adversarial learning process simultaneously trains two neural networks: a generator, which aims to replicate the data distribution, and a discriminator, which distinguishes between authentic data samples and those generated by the model. The objective of the generator is to enhance its outputs to deceive the discriminator. This dynamic interaction results in a minimax optimization scenario that forces the generator to produce increasingly realistic data samples until the discriminator can no longer differentiate them.

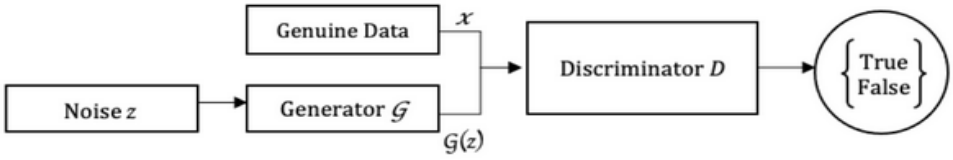


Figure 2. Vanilla GANs architecture, sourced from Dunmore et al., (2023)

Renjith et al. (2021) proposed a methodology to evade malware detection utilizing a pre-trained Generative Adversarial Network (GAN) model to generate an evasive feature vector that misleads malware classifiers into categorizing malicious applications as benign. Rafiq et al. (2022) proposed an adversarial training scheme to enhance the detection accuracy of machine learning (ML) classifiers against adversarial evasion attacks in Android malware detection. TrickDroid combines adversarial data generated through Oracle-based attacks and GAN-based attacks to address the vulnerabilities inherent in both linear and ensemble classifiers, demonstrating resilience against Oracle-based and GAN-

based attacks. This research was proposed to improve Android malware detectors using GAN capabilities. However, this approach can generate novel SQL injection (SQLi) attack payloads for their offensive use in the security testing of firewalls that circumvent firewall protections. This methodology is also applicable in real-world scenarios, in contrast to reinforcement learning (RL).

Therefore, Lu et al. (2022) took a novel approach to ensure the generated samples are not only diverse but also semantically valid by utilizing a combination of GAN and GA. It improved training for detection systems. The objective of the study was to generate datasets that can be used to train machine learning models to detect increasingly sophisticated cyberattacks. The publicly available datasets are highly anonymized and lack standardization and diversity, which restricts their ability to provide a comprehensive and realistic representation for effective training of detection systems.

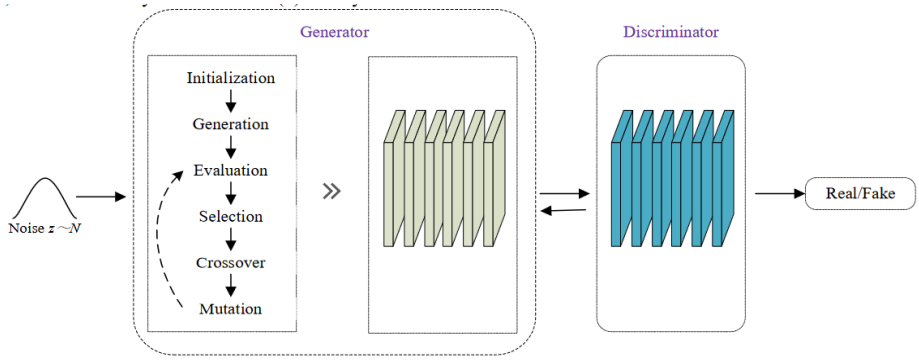


Figure 3. GANs and GA based SQLi Attack Sample Generation Method, derived from Lu et al. (2022)

Therefore, the hybrid framework shown in Figure 3 lays the foundation and contributes to the development of advanced cybersecurity defense mechanisms and solutions by improving datasets. Genetic Algorithms (GAs) enable the generation of

complex, diverse, and high-quality SQL injection payloads capable of bypassing firewalls. This approach has the potential for advancing cybersecurity research and enhancing defensive mechanisms, and the practicality of this approach is demonstrated by the fact that the generated SQL injection samples successfully bypassed Web Application Firewalls (WAFs), such as SafeDog, and validated their efficacy in controlled environments. Incorporating genetic algorithms enhances sample variability through crossover and mutation, while the adversarial training of GANs ensures the authenticity and realism of the generated payloads. However, this study was designed to improve the capabilities of AI-driven intrusion detection systems against sophisticated and advanced SQLi attacks rather than directly employing them in the exploitation phase. The novelty and semantic correctness of generated payloads were not addressed.

However, this research could enhance the attack payloads generated by Transformer models in DeepSQLi to improve its capability against firewall-protected applications. The GAN and GA based hybrid model provides a foundational framework for this praxis research to generate advanced attack payloads for real-world applications to evaluate the security of firewalls. Furthermore, security teams can adopt this approach to assess firewalls and defense mechanisms during testing by integrating them into existing tools to leverage their capabilities and streamline the testing process.

A similar approach was adopted by Chowdhary et al. (2023) to generate XSS attack samples that have the capability to bypass firewalls in order to train IDS systems. The reinforcement learning-based framework was designed to generate and evaluate attack payloads, specifically focusing on sequences of tokens that could represent potential malicious inputs.

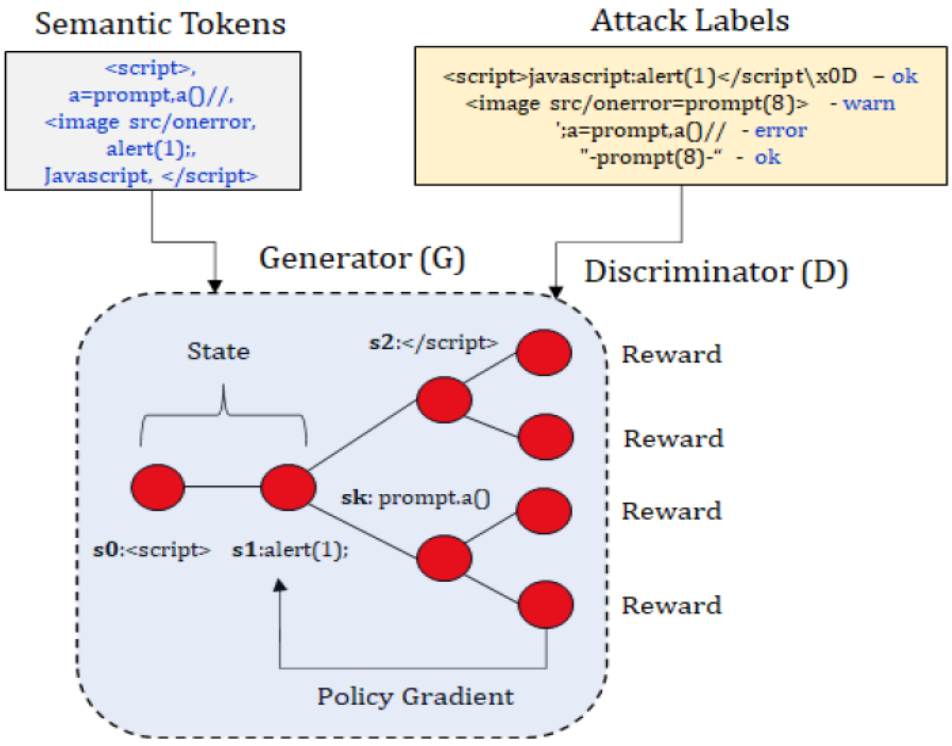


Figure 4. Reinforcement Learning based Framework for XSS Attack Generation, based on Chowdhary et al. (2023)

The system exhibited in Figure 4 comprises two primary components: the Generator (G) and the Discriminator (D), which interact iteratively to refine the detection and generation process. Through multiple iterations, the Generator enhances its ability to produce more realistic, meaningful attack payloads or identify potential vulnerabilities more effectively. Concurrently, the Discriminator improves its capacity to distinguish between safe, suspicious, and malicious sequences. This research mitigates the effort invested in manual penetration testing and identifies attack patterns beyond traditional signature-based attack detection. This technique demonstrates particular efficacy in detecting sophisticated attacks during penetration testing that can circumvent firewalls. However, Reinforcement Learning (RL) necessitates substantial time to learn from the environment, which does not address the time constraints of penetration testing.

Consequently, the complex nature of the real-world application of this approach renders it impractical to implement in real-world applications (Becker et al., 2024).

Boekweg (2024) proposed an SQL Injection Exploitation Tool based on Natural Language Generation. The proposed method produced SQL injection payloads to create a dataset of SQL payloads by utilizing a GAN-based Natural Language Generator (NLG). This tool surpassed SQLMap, OWASP ZAP, and Nikto in terms of accuracy and exploitation by successfully exploiting all tested web pages. While the efficiency of the tool is comparable to existing tools in terms of runtime, it provides more effective results, thereby increasing its efficacy in identifying and exploiting SQL injection vulnerabilities. However, the payloads generated during this research were duplicates of existing queries in the dataset and did not produce genuinely novel attack payloads. Furthermore, the tool is incapable of mutating the specific payload that can be used to test the resilience of the firewall. The real-world applicability of the tool requires validation. Qu et al. (2024) focused primarily on WAF bypass and payload validity. Neither novelty nor diversity was measured nor incorporated into a reward system to guide the model in balancing exploration and evasion objectives.

2.5 Summary and Conclusion

During this literature review, it was observed that traditional tools, such as SQLmap, Acunetix, Burp Suite, and Nikto, and penetration testing approaches utilizing combinations of tools are effective and practical in identifying and exploiting SQL injection (SQLi) attacks in real-world applications. These tools assist penetration testers and security teams by automating time-consuming and tedious tasks by minimizing manual efforts. As a result, the testing duration is significantly reduced, which contributes to

improving the overall penetration testing process. However, these tools are based on fuzzing and predefined payloads, due to which they are incapable of generating attack payloads that are novel, diverse, semantically correct, and WAF-evading at the same time. Also, these tools lack effectiveness against advanced defense mechanisms such as WAFs. On the other hand, machine learning (ML)-based methods were able to generate novel attack payloads. But their effectiveness was never measured based on novelty, diversity, validity, semantic correctness, and WAF bypass simultaneously analyzing their effectiveness against real-world applications, such as how novelty and diversity contribute to the structure of the generated payloads and how WAF bypass is influenced by these dimensions. The existing solutions lack novelty, diversity, validity, semantic correctness, and WAF bypass capabilities in a balanced combination required for complex payload generation. Therefore, this research overcomes the limitations of existing solutions by generating unseen, diverse, valid, and evasive payloads and evaluating them on an evaluation metric proposed for this research for SQLi payload generation models to see their effectiveness and applicability.

Chapter 3—Methodology

3.1 Introduction

The methodology chapter is divided into various sections. The first section provides a high-level and architectural overview of the research methodology and defines the clear path followed during the research. The next section is aimed at data collection and model design to address the limitations of existing SQLi exploitation solutions identified during the extensive literature review in the last chapter. Then, the architecture of models and hyperparameters is discussed. Later, the chapter discusses how the output of the models is deployed against the test environment and the metric used to evaluate the performance of the developed models to figure out the most suitable approach to generate novel attack payloads. The results are evaluated based on a defined metric. The last section highlights the limitations and constraints of the methodology proposed for this research.

3.1.1 Overview of Methodology

The foundation of the research methodology was based on a literature review to find the gaps and limitations of existing tools and solutions discussed in Chapter 2. To fill the gaps and to overcome the limitations, two types of machine learning approaches were proposed for this research. The first ones were standalone TextGAN and stacked LSTM sequential-based models, solely learning from training datasets without any feedback. The second architecture of models was guided by novelty, diversity, validity, semantic correctness, and WAF bypass to direct the model to generate SQLi payloads. Then, the performance of both types of models was evaluated, along with existing research and studies, to analyze their effectiveness based on the evaluation metric. The novelty and diversity of generated payloads were measured by calculating distances between the dataset

and the generated payloads. Semantic correctness and WAF bypass were measured by deploying payloads on a web application using Burp Suite Intruder functionality. The methodology was designed to evaluate research questions and hypotheses. The output of these models in the form of generated payloads yielded novel and complex payloads that can be deployed in real-world settings. Figure 5 presents a high-level overview of the research methodology.

Figure 5. High-Level Overview of Research Methodology

3.2 Data Collection

3.2.1 Dataset and Preprocessing

Publicly available SQLi payload data is acquired from online resources to train these models. The data was collected from Kaggle and GitHub, which consists of known SQL injection payloads. The dataset consists of 30,000 attack payloads of different types of SQLi, including Boolean-based, time-based, and error-based SQLi attacks. The data quality was improved by adding manually crafted payloads to the test environment. This dataset trains TextGANs and stacked LSTMs standalone models to generate novel SQLi attack payloads to exploit SQLi vulnerabilities during penetration testing. Each payload was preprocessed into a sequence of discrete tokens using a custom-designed security-aware tokenizer, which extracted SQL keywords, operators, control characters, custom placeholders, and numeric constants. After tokenization, a vocabulary was constructed

from the unique tokens. Each token was mapped to a unique integer ID. Padding tokens were added to pad the sequences to the maximum observed length.

3.3 Model Design

3.3.1 - TextGAN Architecture for SQLi Attack Payloads

The TextGAN model used to generate SQLi payload consists of two neural networks, a Generator and a Discriminator, as shown in Figure 6. The model was trained on a dataset of payloads collected.

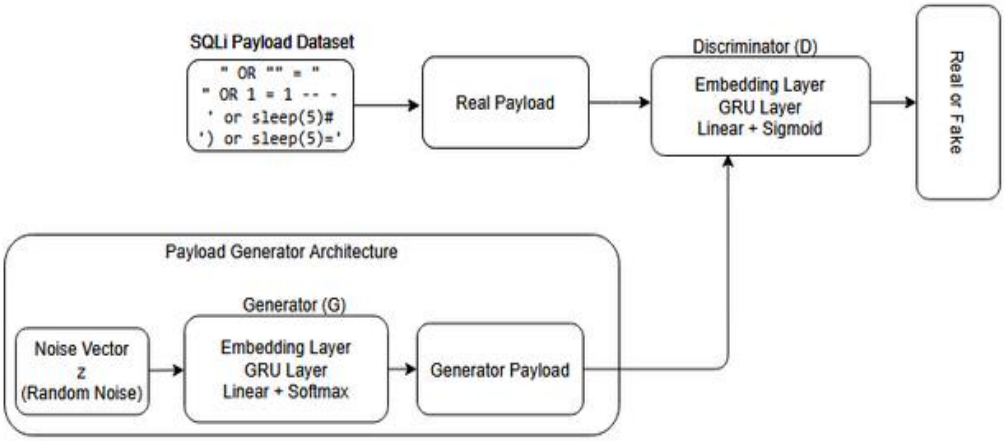


Figure 6. GAN architecture consists of a generator and a discriminator

The generator begins with a noise vector $z \sim N(0, I)$ sampled from a multivariate normal distribution. At each time step t, the input passes through an embedding layer and a GRU to produce the hidden state $h_t^G = GRU_G(Embe\,d(z_t), h_{t-1}^G)$. This hidden state is used to compute token probabilities via a SoftMax layer $y_t = Softmax(W_G h_t^G + b_G)$. A token is then sampled from the resulting distribution $\hat{x}_t \sim y_t$. The final output of the generator is a sequence of sampled tokens $G(z) = \{\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_T\}$. The generator hyperparameters are listed in Table 2.

Discriminator would take each token x_t from real input data, taken directly from your dataset in the message, and run it through the Embedding spell $e_t = Embe\,d(x_t)$. With the embedded token $\boldsymbol{e_t}$ in hand, D passed it into its memory-imbued heart—a Gated Recurrent Unit (GRU). The GRU remembered past tokens and updated its hidden state $h_t^D = GRU_D(e_t, h_{t-1}^D)$ accordingly. This way, D understood not just the current word, but the whole story so far. At the end of the sequence, with all the knowledge collected in the final hidden state h_t^D D made its judgment. It applied a linear transformation $D(x) = \sigma(W_D h_T^D + b_D)$ followed by a sigmoid gate to produce probability. Here, σ was the sigmoid function, gently squeezing the result into a range between 0 (fake) and 1 (real).

Table 2. Generator Hyperparameters

Hyperparameter	Value	Description
latent_dim	64	Dimension of the input noise vector (z).
hidden_units_1	128	Hidden size of the GRU layer used to model sequential structure.
activation	tanh, sigmoid	GRU uses internal tanh and sigmoid activations for gating/state update.
output_activation	Sigmoid	Activation function for output layer to produce token probabilities from output logits.

The discriminator network takes as input either real or generated SQL payloads and determines their authenticity. It begins with an embedding layer that maps token IDs to dense vectors. These embeddings are processed by a GRU layer, which captures the sequential dependencies in the payload. The final hidden state is passed through a linear layer followed by a Sigmoid activation function, producing a probability score between 0 (fake) and 1 (real). This RNN-based architecture allows the Discriminator to model the

structure and syntax of SQL payloads effectively. The generator and discriminator continuously improve themself over time. The generator generates increasingly realistic attack payloads while the discriminator improves itself in differentiating between real and synthetic SQLI attack payloads.

3.3.2 Adversarial Training Loop for GANs

Since the model was trained using standard adversarial training with a batch size of 32 and an initial learning rate of 0.001 for both the Generator and Discriminator. The Generator was first pretrained for 50 epochs using a language modeling objective. Adversarial training continued for 1000 epochs, with the Discriminator updated every 5 epochs to avoid destabilizing training. Embedding and hidden dimensions were set to 64 and 128, respectively. Temperature-controlled sampling (0.9–1.1) was used to balance diversity and coherence in generated payloads. The model training hyperparameters and optimization parameters are provided in Tables 3 and 4, respectively.

The model's training process involves a generator and a discriminator, and both of these models compete with each other. The model undergoes training by successively updating the discriminator and generator (Huang et al., 2024). First, the discriminator is trained by selecting a batch of real SQLi payloads and a batch of fake ones using the generator. The discriminator then learns to classify these payloads correctly, improving its ability to differentiate between authentic SQLi attempts and artificially generated ones. Next, the generator is trained by sampling random noise vectors from a normal distribution and using them to generate synthetic SQLi payload feature vectors. The goal of the generator is to produce SQLi payloads that are indistinguishable from real attacks that are real enough to fool the discriminator.

Table 3. Hyperparameters for Training

Hyperparameter	Value	Description
Epochs	1000	Number of training iterations over the dataset.
Batch Size	32	Number of samples per batch during training.
Discriminator Training Frequency	5	Number of samples per batch during training.
Label Smoothing	0.9	Real label value for discriminator loss.
Pad Token ID	10	Padding token index used during sequence padding.

Table 4. Model Optimization Parameters

Parameter	Value	Description
g_optim.lr	0.001	Learning rate for generator (Adam optimizer)
d_optim.lr	0.001	Learning rate for discriminator (Adam optimizer)
Loss function	Binary Cross-Entropy	Used for both generator and discriminator training

3.3.3 Generating Novel SQLi Payloads

A fully trained model generated a novel SQLi attack payload. The generation process was initiated by generating random noise vectors, which were then transformed into synthetic SQLi payload feature vectors. The generated payloads were novel and complex, which overcame the limitation of predefined script-based SQLi tools and techniques, and were deployable for the exploitation of SQLi in penetration testing.

3.4 – Stacked LSTM- A Sequence-Based Generator for SQLi Payloads

In addition to the GAN-based approach, another standalone stacked LSTM model was used to generate SQL injection (SQLi) payloads to analyze their learning capabilities for SQLi attack payloads. This model was designed to learn sequential token patterns from the training dataset using a language modeling framework. Unlike GANs, the Stacked LSTM directly learns the conditional probability distribution of tokens given prior context.

Generating novel SQLi payloads using Long Short-Term Memory (LSTM) networks followed a structured data preprocessing, model training, and generation. The goal was to train an LSTM-based sequence model capable of learning patterns in existing SQLi attack payloads and generating realistic, novel attack strings that can be integrated into penetration testing.

3.4.1 Model Architecture

The architecture of the stacked LSTM model was made up of an embedding layer, two LSTM layers, and a linear projection layer, as shown in Figure 7. The embedding layer maps tokens into a continuous vector space, while two LSTM layers are used to capture long-range dependencies in payloads. The linear projection layer transforms LSTM outputs into vocabulary logits.

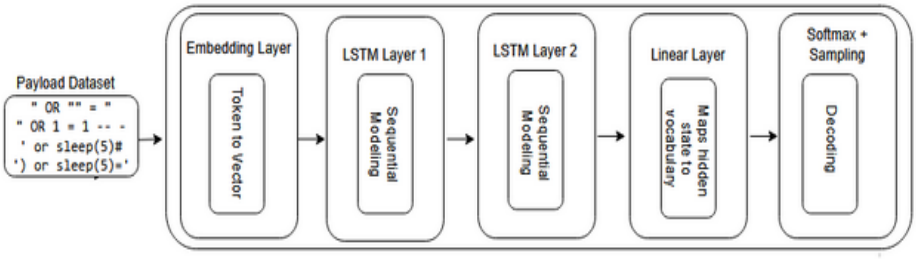


Figure 7. LSTM architecture for SQLi Payload

The model was designed to generate SQL payloads for testing web application security. The model learned SQLi attack payloads to produce SQL injection strings by analyzing patterns from a dataset of known payloads. The input of the model was a sequence of tokens from a SQL injection payload. These tokens $\{x_1, x_2, \ldots, x_T\}$ were fed sequentially into the model, one at a time. Each input token was converted to a dense vector $e_t = Embedding(x_t)$. This embedding captured the syntactic and semantic features of SQL elements. The embedded sequence was then processed by two stacked LSTM layers to capture the structure and dependencies in SQL payloads:

$$h_t^{(1)}, c_t^{(1)} = LSTM_1\left(e_t, h_{t-1}^{(1)}, c_{t-1}^{(1)}\right)$$

$$h_t^{(2)}, c_t^{(2)} = LSTM_2\left(h_t^{(1)}, h_{t-1}^{(2)}, c_{t-1}^{(2)}\right)$$

The layers allow the model to remember the context of SQLi payloads, for instance, that after ' OR, a condition like 1=1 usually follows. The final LSTM output was passed to a linear layer that maps it to the vocabulary of possible tokens $z_t = Wh_t^{(2)} + b$.

Then SoftMax $y_t = Soft\,max(z_t)$ gave a probability distribution over the next possible token. So, if the current sequence was ' OR, the model might assign high probability to tokens like 1, username, or sleep(5). A token was sampled from the probability distribution y_t and used as the next input $\hat{x}_{t+1} \sim y_t$. This model learned SQL injection patterns from training data and then generates payloads token by token. The LSTM layers enabled the model to maintain context across multiple tokens, while SoftMax sampling introduces diversity in the generation process. The model hyperparameters used during the training are provided in Table 5.

Table 5. Network Architecture Parameters

Hyperparameter	Value	Description
Embedding Dimension	128	Size of token embedding vectors
LSTM Layers	2	Number of stacked LSTM layers to learning sequential patterns in SQLi payloads
Hidden Dimension	128	Number of units in each LSTM layer
Sequence Length	Variable	Maximum tokenized payload length
Batch Size	32	Number of samples per batch
Epoch	100	Total training iterations over data
Learning Rate	0.001	Adam optimizer learning rate
Loss Function	CrossEntropyLoss	Token-wise classification loss
Sampling Temperature	1.0	Controls randomness during generation

The model was compiled using the categorical cross-entropy loss function, as this is a multi-class classification to choose the correct next character from multiple possibilities. The Adam optimizer was used to optimize learning.

3.4.2 Model Training

The Stacked LSTM model was trained with an embedding dimension and hidden size of 128, using two LSTM layers to capture token-level dependencies. Training was performed over 100 epochs with a batch size of 32 and a learning rate of 0.001 using the Adam optimizer. Padding tokens were handled explicitly using a reserved index beyond the vocabulary range, and no dropout was used. The sequence generation process applied temperature-controlled sampling with $T = 1.0$

3.4.3 SQLi Payload Generation

When the model was trained, an initial seed query was provided as input to generate novel SQL Injection payloads. The process started by tokenizing and padding the seed text, such as "1' OR ", to match the required input length for the model. The trained model then made predictions for the next character in the sequence by producing a probability distribution for possible next tokens. Furthermore, a randomized sampling technique called temperature scaling was applied to balance predictability and novelty in the generated payloads to ensure that the model did not always choose the most likely token to allow for variability or diversity in the output.

3. 5 Evaluation Metric Embedded RL-LSTM Model

This novel approach was based on a feedback loop that consisted of novelty, diversity, validity, and semantic correctness, and WAF bypass compared to the previous standalone model approach. The architectural diagram of the evaluation embedded model is shown in Figure 8. The model initiated its work with a seed payload similar to a human penetration tester, who injects an SQLi payload into a vulnerable input field and analyzes the response of the web application.

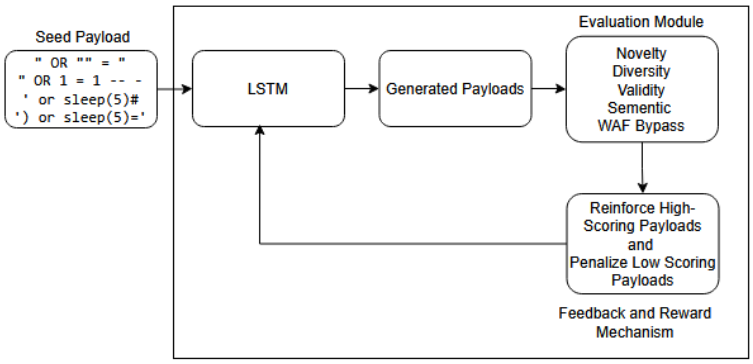


Figure 8. Evaluation Embedded RL - LSTM architecture for SQLi Payloads

In this method, an LSTM model was used along with RL and an evaluation metric. Then, the mutator function in the tool mutates payloads by applying different mutation strategies. Then the mutated payload was evaluated by the evaluation function, which consists of the evaluation metric. The total reward R assigned to a generated payload was a weighted sum of all combinations of five evaluation components of novelty, diversity, validity, semantic, and WAF bypass.

$$R = a_1 \cdot Valid + a_2 \cdot Bypass + a_3 \cdot Novelty + a_4 \cdot Diversity + a_5 \cdot Semantic$$

Where a_i are the weights assigned to each dimension.

Here, novelty ensured that the generated payload was different from the previous one by calculating the cosine distance from the seed payload. Diversity ensured that the payload was written in different forms, such as SELECT as SeLeCt and sElEcT, etc. The overall form of the generated payload was modified by applying different mutation techniques. SQLparser ensured that the payload was syntactically correct, which means it follows the SQL language syntax. While semantic correctness ensured that the payload was executable and performed tasks on the database.

3.6. Experimental Setup

The output of the models generated in the form of novel and diverse payloads was evaluated in the test environment for semantic correctness and WAF bypass. Therefore, a simulated test environment shown in Figure 9 was set up to deploy and test the payloads generated by the standalone and feedback-based models. The experimental setup consisted of two components of deployment and testing. The following diagram represents the environment that serves as a testbed for the evaluation of research hypotheses.

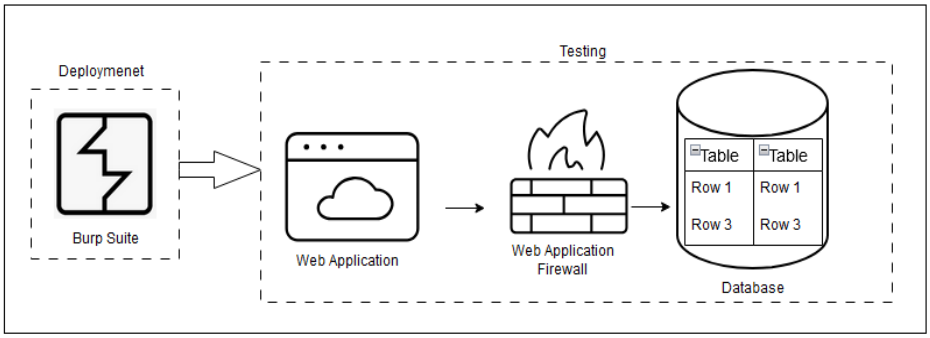


Figure 9. Environment Setup - Deployment and Testing Phase of Methodology

3.5.1 - Deployment

The generated SQLi payloads were deployed for practical applications on vulnerable applications to analyze their semantic correctness and WAF bypass. Burp Suite was leveraged to achieve deployment and to make payloads applicable to real-world testing. The deployment phase was carried out with Burp Suite in a simulated environment by loading novel attack payloads into the tools and injecting them into the vulnerable fields to evaluate the performance of models based on the evaluation metric. Furthermore, it was

deployed to integrate novel attack payloads to vulnerable endpoints of web applications due to its crawling capabilities. In addition, session handling was managed through this tool to test endpoints accessible to authorized users. The generated payloads were imported as payloads to Burp Suite Intruder and injected automatically using Burp Suite's Intruder functionality.

3.5.2 - Testing

The testing environment consisted of a database-driven web application protected by WAF rules. The application with a database served as a public-facing web application to simulate real-world environments containing SQLi vulnerabilities. A vulnerable web application was used as an application with injection endpoints that contain various types of vulnerabilities. This application is deliberately designed to be vulnerable for security testers and researchers to evaluate their tools and techniques. This research was focused on input fields vulnerable to SQLi vulnerabilities to inject attack payloads generated by GANs and LSTM models to analyze their effectiveness in exploitation. The bWAPP instance was hosted on a local machine disconnected from external networks to ensure that all SQLi tests are conducted in a controlled and safe environment.

WAF rules served as a simulated firewall that was deployed to stop the injection attacks performed on web applications. The purpose of deploying a simulated firewall was to evaluate the effectiveness of the novel attack payloads to see whether they were novel enough to bypass the firewall detection capabilities compared to existing solutions. In addition, the MariaDB database was configured to contain sample tables that store structured data that were intentionally vulnerable to SQLi attacks. These schemas were

intentionally poorly designed and lack input validation and parameterized queries, so attack payloads can be evaluated for testing purposes.

3.7 - Evaluation Metric

An evaluation metric was proposed for this research to evaluate the performance of models for generating SQLi payloads. This evaluation metric consists of five dimensions of novelty, diversity, validity, semantic success, and WAF bypass, as provided in Table 6.

Table 6. Evaluation Metric for SQLi Payload Generation

Dimension	Method
Novelty	Cosine Distance
Diversity	Token Entropy, 3-gram Diversity, Avg. Levenshtein Distance, Avg. Jaccard Distance
Validity	SQLparser
Semantic Analysis	Executed on a Web Application
WAF Bypass	Rule-based simulated WAF to prevent SQLi attacks.

This evaluation metric was used as a foundational work that integrates all the mentioned dimensions while maintaining a balance to make generated payloads effective and real-world applicable for discovering complex SQLi vulnerabilities. Also, this framework directed the development of models for SQLi payload generation and exploitation.

Novelty of generated payloads refers to the degree to which generated SQLi payloads differ from those in the training dataset. The novelty of the payload was measured by the cosine distance between the generated and training payloads. Diversity assesses the lexical and structural variety across generated payloads. The diversity of the generated payloads was calculated using a number of structural diversity metrics, such as token entropy, n-gram diversity, and average Levenstein/Jaccard distances

between generated SQLi payloads beyond surface-level novelty. These metrics provide information on how diverse the model's outputs are, as well as whether they show syntactic variety or repeat patterns. The validity of the SQLi payloads was evaluated through an SQL parser as a baseline to validate the grammar and syntax of the generated payloads. This method was useful for measuring the payloads generated by the models without a feedback loop.

The generated valid attack payloads were further tested for semantic correctness by applying them to a web application. This was useful for further separating the broken and semantically incorrect payloads beyond validity checks. Therefore, semantic success was achieved by executing generated payloads on the test environment to see how many of the payloads were executable on the database. Are they extracting data from the database tables? Finally, the WAF bypass was measured to determine how many novel, diverse, and valid payloads were able to evade the firewall. This was calculated by applying payloads to the web application protected by WAF.

3.7 Limitations

The proposed approach had several limitations and challenges. First was TextGANs' training instability because the architecture of the model was standalone, without any direction, and it was unable to generate valid and syntactically correct payloads when trained on the dataset. Generative Adversarial Networks (GANs) learned patterns in data but did not inherently understand the underlying logic behind those patterns. In the context of SQL injection (SQLi) payload generation, a TextGAN produced payloads that were syntactically valid but logically meaningless because it lacked explicit knowledge of SQL grammar rules, database structures, and security mechanisms.

Furthermore, there was no direct integration of the model with the test environment, which can impact the semantics of generated payloads. The direct integration can enhance the performance of the model, but it requires significant computational resources, and real-time validation against WAFs can introduce additional overhead. Training deep learning models for AI-generated SQLi payloads was computationally expensive, requiring high-performance GPUs/TPUs, extensive storage, and large-scale dataset processing. However, this issue was mitigated by deploying the payloads with Burp Suite. Lastly, using AI-generated novel SQL injection (SQLi) attack payloads presented significant ethical and legal concerns, mainly when deployed without authorization. While generating new and evasive SQLi payloads aimed to improve penetration testing and strengthen cybersecurity defenses, these payloads can violate cybersecurity laws if tested on unauthorized systems. Therefore, testing was conducted in a controlled environment to prevent potential misuse and legal repercussions.

3.8 Conclusion

The methodology provides a comprehensive approach to SQLi payload generation, including a literature review, model design, evaluation metric, and environment setup for deploying payloads in a controlled way. The literature review provided a starting point to identify the limitations in existing SQLi exploitation solutions, and it showed that existing solutions were based on predefined test cases and were unable to generate novel attack payloads. Therefore, TextGANs, stacked LSTMs, and feedback models guided by an evaluation metric were proposed and developed to overcome the limitations by generating novel, diverse, valid, semantically correct, and WAF-bypassing attack payloads. Additionally, SQLi payloads produced by GAN and LSTM models were assessed and

examined using a predetermined evaluation metric. The evaluation was carried out using the payload's adversarial efficacy against a simulated environment. The test environment served as a model evaluation and refining testbed and guaranteed safe testing. Models were further refined based on the evaluation metric to optimize the performance of GANs and LSTMs to generate novel attack payloads.

Chapter 4—Results

4.1 Introduction

The goal of this research was to evaluate existing SQLi exploitation solutions based on the proposed evaluation metric and to develop SQLi payload-generating models using the same evaluation framework for improved performance and practical utility. The proposed framework was discussed in detail in Chapter 3, which consists of five dimensions: novelty, diversity, validity, semantic success, and WAF bypass. All these critical components were required to identify complex and undiscovered SQLi vulnerabilities in web applications. Therefore, an evaluation metric was used as a framework for assessment and improvement based on the strengths and weaknesses of existing SQLi exploitation solutions.

An exploratory approach was adopted to identify the limitations of existing solutions through a literature review and by asking the given questions: Are existing tools and AI models generating novel attack payloads? Are these payloads diverse? Can these payloads be valid and semantically correct? Can they bypass WAFs during testing? These questions guided the development of an evaluation framework for assessing existing methods and directing the advancement of the model.

This chapter is divided into two main sections: the first section presents the results for Research Questions 1 and 2, while the second section addresses Research Questions 3 and 4. To the best of our knowledge, this is the first work that focuses on both the generation and evaluation of payloads produced by models. Thus, we introduced a comprehensive evaluation metric that considers all critical components of payload generation to guide the development of models that are both balanced and effective.

While other studies focus on specific aspects of SQLi exploitation, such as WAF bypass, some highlight the need for novel payloads to exploit complex and previously unknown vulnerabilities. The novelty of SQLi payloads was addressed by DeepSQLi. AdvSQLi focused on validity and WAF bypass but overlooked novelty. DCGAN employed a similar approach to AdvSQLi, generating payloads to expand datasets and bypass WAFs. However, these existing methods only assessed novelty and WAF bypass in isolation. None of the previously proposed solutions addressed all five dimensions simultaneously while maintaining the balance, including novelty, diversity, validity, semantic success, and WAF bypass. Therefore, this study was focused on bridging the gap by developing a model that balances these components for more effective exploitation of SQLi vulnerabilities. Accordingly, the proposed comprehensive evaluation metric was used to measure the novelty, diversity, validity, and effectiveness of generated payloads and to thoroughly evaluate the performance of developed models.

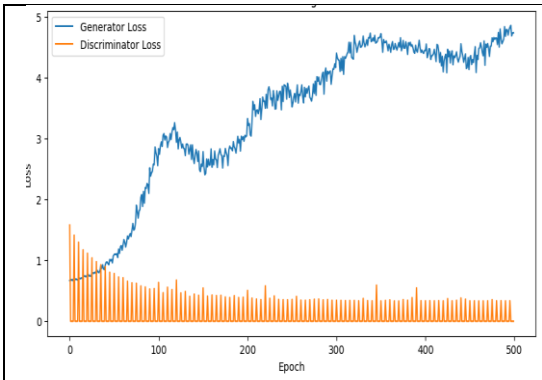
4.2 TextGAN and Stacked LSTM Model Training Performance

In this approach, two standalone models, TextGAN and LSTM, were trained on the dataset of payloads, so that their training behavior and output in the form of generated payloads could be analyzed based on the defined evaluation metric. The following section provides insights into the training behavior of TextGAN and stacked LSTM models on the SQLI attack payloads dataset. The training graph of both standalone models showed how each of them performed during adversarial training, offering both practical utility and theoretical novelty. The objective of training analysis was to evaluate the training stability, convergence behavior, and learning dynamics of each model through loss curves and based on the evaluation metric. Both models were trained on the dataset without any feedback

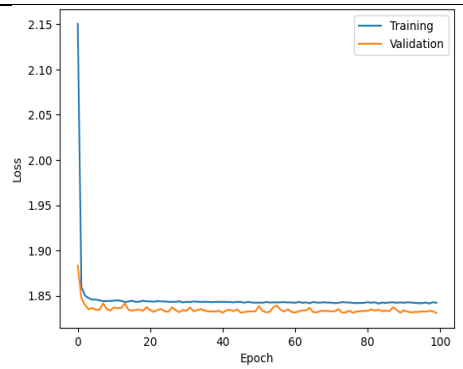
loop to analyze how effective the generated payloads were when it comes to novelty, diversity, validity, semantic correctness, and WAF bypass.

4.2.1 Training Behavior of TextGAN and Stacked LSTM

As mentioned earlier, the models were trained on the same dataset to generate novel SQLi attack payloads. During the training of the TextGAN, it was observed that the Generator loss rises initially in the early stages while the discriminator loss rapidly decreases and stabilizes at a low value. On the other hand, the generator loss increased steadily with oscillations and periodic dips. The decrease in lower loss occurred at moments when the Generator temporarily fooled the discriminator effectively. The training loss curves indicate unstable adversarial training between the Generator and Discriminator due to a stronger discriminator that was effective at distinguishing real and fake payloads. The training behavior was diverging, and the generator loss did not converge to a fixed minimum.



TextGAN Generator and Discriminator Loss Curve Over 500 Epochs



LSTM Training Loss Curve Over 100 Epochs

Figure 10. Training Curves of TextGAN and Stacked LSTM

To overcome the issue of training instability and divergence, several techniques, such as reducing the training frequency of the discriminator, label smoothing, and introducing

noise, were applied throughout the development process. First of all, the approach was to reduce the frequency at which the discriminator was trained. The discriminator was updated every five epochs instead of being updated on every epoch to prevent the discriminator from learning too quickly and overwhelming the generator. Secondly, label smoothing techniques were used to slightly lower the value of labeling real data from 1 to 0.9 to encourage the discriminator to remain uncertain and less confident. In addition, a small amount of noise was introduced into the real input data to force the discriminator to generalize better and reduce the risk of it memorizing the real data too easily. Also, the generator was trained in a supervised manner for the first 50 epochs to give a strong start. The purpose of doing this was to enable the generator to learn the basic structure of the data. The training of the model was monitored during the entire training process. Although all of the above-mentioned stabilization techniques were used to balance the training between the generator and the discriminator, the issue of instability persisted. The instability was visible in the form of increasing generator loss and very low discriminator loss. Therefore, the training graph of TextGAN points to an imbalance between the discriminator and the generator because when the discriminator becomes too good at differentiating between real and fake payloads, it leaves the generator with little useful feedback to learn from.

On the right side of Figure 10 is a stacked LSTM model training graph, which consists of training and validation loss. The model was trained in over 100 epochs. The graph shows a rapid loss in early epochs, and both lines show a sharp decrease. This behavior indicates that the model has learned payload patterns effectively during the initial epochs. After almost 10 epochs, the model achieved a convergence phase, which was a clear indication

that the model had learned almost all patterns in the dataset. Most importantly, there were no signs of divergence and oscillation during stacked LSTM training. The absence of divergence behavior demonstrates that the model architecture played a vital role in training stability as compared to TextGAN due to its sequence learning capabilities.

4.2.1 Evaluation of Generated Payloads Based on Defined Metric

After the completion of the training process, the generated payloads by the TextGAN and stacked LSTM were analyzed based on a defined evaluation metric to measure the novelty, diversity, validity, semantic correctness, and WAF evading capabilities of newly unseen SQLi payloads. The novelty of each generated payload was measured by calculating the cosine distance between the generated payloads and the training dataset. The payloads that exhibited a cosine distance greater than 0.5 were considered highly novel because of their structural differences from the training dataset. It was observed during the analysis that out of 100 payloads generated, only 5 of them had a cosine distance greater than the 0.5 threshold. TextGAN produced more highly novel payloads compared to the stacked LSTM, as shown in Table 7. The center of the distribution was between 0.38 and 0.45, which means that most of the payloads were moderately novel. The novelty distribution produced by both models is shown in Figure 11.

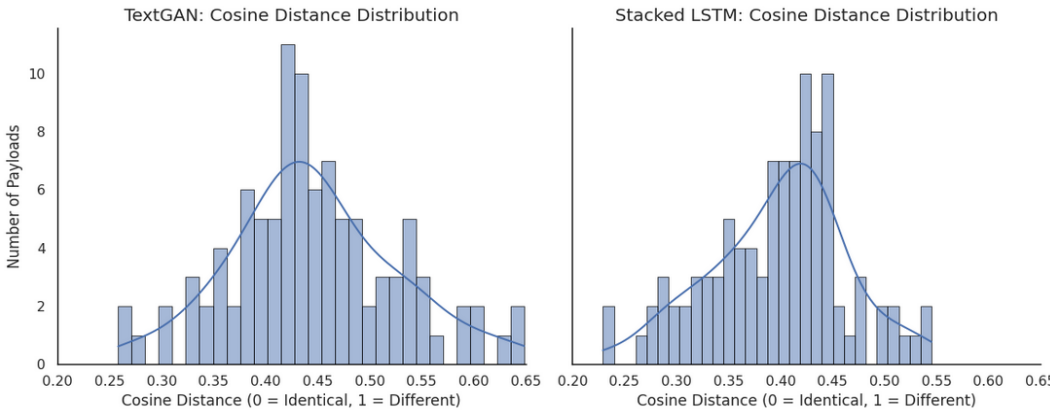


Figure 11. Novelty of Generated Payloads

In Figure 11, the x-axis is the cosine distance, where 0 indicates that generated payloads were the same as their training dataset payloads, while 1 indicates that payloads were completely different. On the y-axis, there is the count of produced payloads for each distance bin. The distribution peaks around showed that most of the generated payloads are moderately different from the training payloads. While only a small number of payloads have a distance < 0.2, this indicates a sign of memorization. Novelty cosine distances were close to 0.6. It was observed during the novelty analysis that TextGAN produced more highly novel payloads compared to the stacked LSTM model. However, the output of both models was not random or totally unrelated to the training data. In other words, the models learned the patterns of SQLi attack payloads, and there was a relation between the generated payloads and the payloads in the training dataset.

Table 7. Novelty comparison of stacked LSTM and TextGAN

Metric	TextGAN	Stacked LSTM
Novelty (Cosine > 0.5)	~12–14%	5.00%

Then the diversity of the generated payloads was calculated across the diversity metric, which consists of token entropy, 3-gram diversity, Levenshtein, and Jaccard distances. Table 8 shows the diversity comparison of models across 100 produced payloads. The table reported the exact metric values for each model, while Figure 12 provides a visual comparison to illustrate the performance differences more clearly.

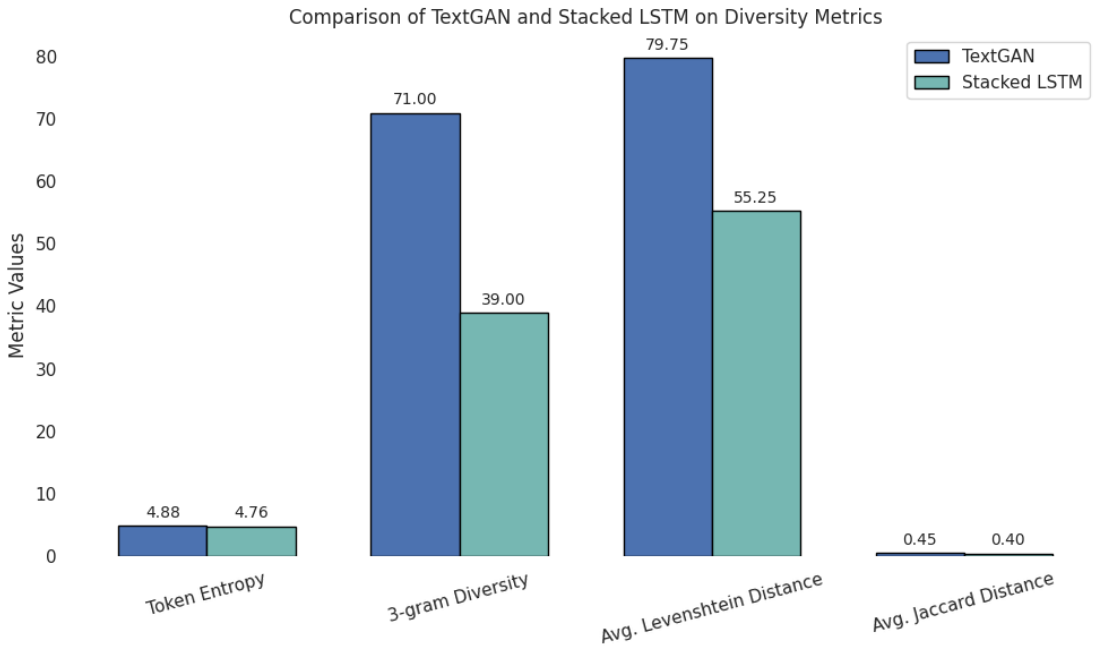


Figure 12. Diversity Metrics

Table 8. Summary of Diversity Measures Applied to Generated SQLi Payload

Metric	TextGAN	Stacked LSTM
Total Payloads	100	100
Uniqueness Ratio (%)	100.00%	100.00%
Token Entropy	4.8751	4.7565
3-gram Diversity	71%	39%
Avg. Levenshtein Distance	79.75	55.25
Avg. Jaccard Distance	0.4460	0.4011

The analysis of diversity in the generated payloads showed that both adopted models exhibited a 100% uniqueness ratio, which means there were no duplicates among the 100 payloads in each dataset. Token entropy showed that both models produced diverse payloads with consistent structure, which was a sign of non-repetitive payloads. However, TextGAN achieved higher 3-gram diversity than LSTM. TextGAN demonstrated more exploratory behavior compared to the stacked LSTM, which was a clear indication of a good amount of local variance in the data's structure. In addition, higher average Levenshtein distance and Jaccard distance were observed in TextGAN compared to the stacked LSTM. This showed that TextGAN generated more diverse payloads than the stacked LSTM.

Furthermore, to check the validity of the generated payloads, a similar approach was adopted to that employed by Lu et al. (2022) to test the syntactic validity of SQL injection samples. Each generated SQL injection (SQLi) payload was tested using a SQL parser to verify its syntactic validity. This served as a basic static analysis step to confirm whether the payloads adhered to the general rules of SQL structure.

Table 9. Parser-Based Validity of Generated SQLi Payloads

Metric	TextGAN	Stacked LSTM
Total Payloads Tested	100	100
Parser-Validated Payloads	100	100
Validity Percentage	100%	100%

Based on this method, both models achieved 100% syntactic validity, as shown in Table 9. Out of 100 generated samples, all were found to be syntactically correct. Although parser-based evaluation showed that both models generated 100% syntactically valid SQLi

payloads, this alone was not sufficient to confirm true validity. This is because parser-based evaluation merely ensures that SQL keywords are present in the generated payloads; therefore, all payloads were able to pass the parser check. However, this does not assess the logical structure or execution behavior of the payloads. To determine whether the models were overfitting or genuinely learning, semantic correctness was evaluated by testing the payloads against a simulated environment. The results of the semantic correctness evaluation are presented in Tables 10 and 11, which show that standalone models, trained on SQL-containing datasets, produced payloads that were syntactically valid but semantically incorrect. In this test, 100 payloads were analyzed and generated by both models on a vulnerable web application using Burp Suite Intruder to analyze their semantics and execution capabilities. Based on this situation, relying on SQLparser accuracy results was not adequate for the validity of the payloads to test the output of the model. A further in-depth analysis was required by deploying payloads on database-driven applications for their execution validation. Therefore, we performed semantic correctness analysis on the web application. Semantic check validation was achieved by deploying generated payloads for execution testing using Burp Suite Intruder on the vulnerable web application. The sample payloads generated by TextGAN are shown in Table 10.

Table 10. Payloads generated by TextGAN

Payload ID	Payload
1	SElECT [randchar] > 5 - -- % WHERE 0 > 1 table_schema + (= # information_schema 0
2	SElECT WHERE benchmark || table_name) (-- + FROM information_schema # [randint] (0 + 1
3	And 23 > (# [randchar] information_schema FROM (schemata FROM --) (table_name) [randchar] tables
4	or -- SELeCT) 1 FROM (# FROM -- table_schema) schema_name -- + FROM -- table_schema

During the semantic analysis and deployment of payloads for execution, it was observed that the model learned SQL keywords and operators effectively. The model was able to substitute different variations of SQL keywords for rule-based WAF bypass. However, these types of basic variations were inadequate for advanced ML-based firewalls. The keyword SELECT was obfuscated as SElECT and SELecT to evade rule-based WAF, as reflected in Tables 10 and 11. The payloads generated by the model include SELECT, WHERE, and FROM, which make up the SQL statement. In addition to that, the model intelligently included a placeholder for dynamic fuzzing. The model generated SQL payloads that resemble real SQLi payloads but did not have the capability to bypass WAF. In this type of standalone architecture, models only learned the SQL keywords and the attack strategy based on training data, such as mutating keywords and constructing new payloads each time.

However, the syntax and semantics of the payload were compromised and were challenging to achieve without any feedback, regardless of the size of the datasets. At this stage, without any feedback, the model only improved the novelty and diversity of payloads, but validity in a real-world environment still required further refinement. On the other hand, stacked LSTM generated meaningful injection patterns, and each payload reflected a unique combination of SQL keywords and obfuscation techniques, while retaining semantic relevance to metadata extraction.

Table 11. Stacked LSTM Generated Payloads

Payload ID	Payloads
1	+ and AND (schema_name (< 5 + # + = database ( ) # #
2	SELECt group_concat % like schemata FROM 1 ' WHERE -- = database ( ) -- --
3	SELECT group_concat DELAY =) FROM + schemata # + = database ( ) -- #
4	% && group_concat [randint] =) FROM information_schema # WHERE < 2 database ( ) -- --

The payloads contained metadata enumeration using schema_name, information_schema, and group_concat, obfuscation through misplaced operators, and a variety of SQLi attack vectors. The inclusion of both standard SQL clauses and dynamic placeholders showed the generative ability of the model to blend known attack patterns with variability. Despite that, none of the generated payloads were semantically correct when applied to real web applications. Because this approach did not directly integrate with the application and models, it just learned patterns from the data. It also highlighted a key limitation of the SQL parser-based evaluation approach used by Lu et al. (2022). Some generated payloads included unused variables, flawed logic, or incomplete fragments that would not execute properly without proper context. Lastly, the practical evasiveness of generated SQLi payloads was tested by determining how well they could get around a certain set of WAF rules. The regular expressions and well-known detection patterns were used to simulate WAF behavior for this research. It was observed during the experiment that payloads generated by both models were not able to evade the simulated rule-based WAF behavior, as shown in Table 12.

Table 12. WAF Bypass Comparison of Stacked LSTM and TextGAN

Metric	TextGAN	Stacked LSTM
Total Payloads Tested	100	100
Bypassed Payloads	0	0

4.3 Comparative Summary of Models with Benchmark Tools

The first research question was evaluated in two ways: The first way was based purely on reported numbers through research studies, because either their code was not available, or they were not compatible with the test environment. Figure 13 above shows the limitations of existing solutions across the evaluation metric based on reported numbers. The second method was based on the experimental evaluation, in which SQLmap was used against a test environment and served as a benchmarking tool to compare the performance of developed models based on the evaluation metric. Figure 14shows how models performed in terms of novelty, diversity, validity, semantic correctness, and WAF bypass during the experiments carried out in a simulated environment.

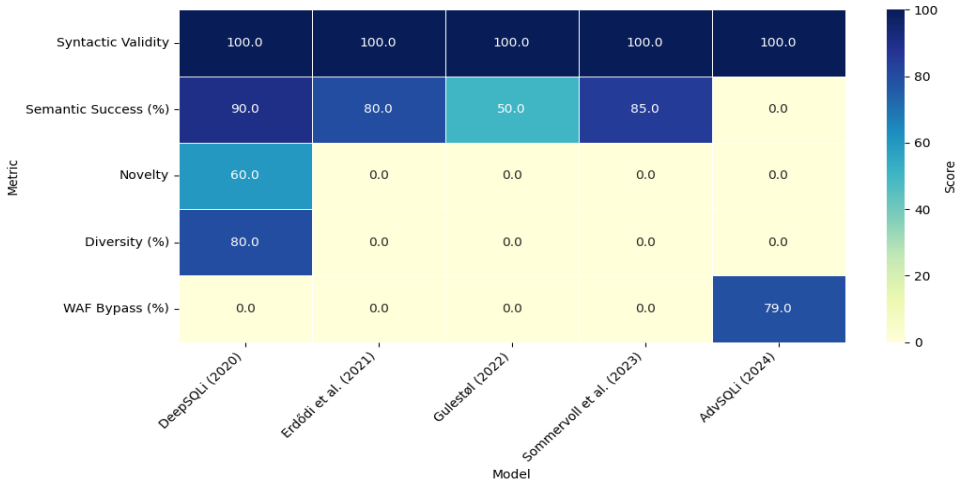


Figure 13. Existing AI solutions evaluated based on Evaluation Framework

The evaluation of existing SQLi exploitation tools and models showed that none of the solutions provides a balanced approach between novelty, diversity, validity, and WAF bypass. The results of the evaluation are presented in Figures 13 and 14. Some of them were heavily focused on WAF bypass and while others focused on novelty while undermining the rest of the dimensions. There was a lack of balance between all these dimensions to ensure that the payloads were functional.

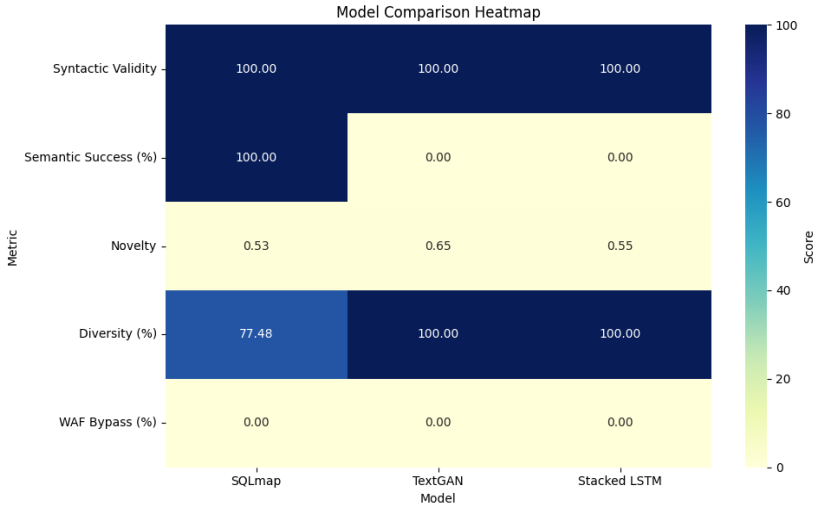


Figure 14. AI models developed to address the limitations of Existing AI solutions, and their performance was compared with the Benchmarking Tool

Figure 14 presents evaluation data obtained from the test environment and calculated based on the evaluation metric. The results were then compared with the benchmark tool SQLmap to analyze the performance of models in terms of novelty, diversity, validity, semantic correctness, and WAF bypass. The results of comparisons showed that TextGAN and stacked LSTM models increased the novelty and diversity of the payloads up to 100% compared to SQLmap. Both standalone proposed models demonstrated high novelty and syntactic validity, but a deeper semantic analysis showed that generated payloads were invalid, ineffective, and functionally broken, and models

failed to produce semantically correct payloads without any direction. On the other hand, SQLmap performed well in terms of semantic success, while its novelty and diversity were lower compared to the machine learning models. Also, none of the models were able to bypass the WAF, including SQLmap.

In conclusion, Stacked LSTM and TextGAN enhanced the novelty, diversity, and syntactic validity of generated payloads when trained on a dataset of payloads. The in-depth analysis performed using quantitative and qualitative analysis showed that both models were excellent at novelty, diversity, and syntactic validity. However, none of the payloads generated were semantically correct by both models.

4.4 Overcoming the Issue of Validity with an Evaluation Metric Feedback

It was observed during the analysis of standalone models, TextGAN and stacked LSTM, that without a feedback loop, both models were able to generate novel and diverse payloads, but they were semantically incorrect and unable to evade WAF. Therefore, to resolve the issue of validity and WAF bypass, an evaluation metric was introduced and embedded as a feedback loop with reinforcement learning. The purpose of this feedback was to provide directions to the model to generate payloads while maintaining a balance between all of the dimensions to keep payloads functional and semantically correct. The main advantage of this approach was that it did not require a huge dataset of payloads, as was required in the last approach. It worked on a single payload and a batch of payloads. This section presents the results for the research questions 3 and 4, where the objective was to see if an evaluation metric-guided approach increases the validity and WAF bypass. Furthermore, how did input payloads impact the novelty, diversity, and WAF bypass performance in reward-based models?

4.4.1 Training Graph of Model

The model was provided with a single payload and a batch of seed payloads to start the process of generating SQLi payloads. When the single seed payload was given, the training fluctuated, and the model showed instability in the reward over the iterations. The reward function was introduced as a summation of novelty, diversity, validity, and WAF bypass. The reward for the WAF bypass was the highest. This exhibits that using a model with an evaluation metric as a feedback loop constrained the model to ensure validity through SQLparser and WAF bypassing through interacting with a regular expression firewall.

The proposed model performed well when provided a batch of seed payloads due to its reward function designed to direct the model to introduce novelty, diversity, validity, and WAF bypass. The model was trained using both a single seed and a batch of seeds to evaluate the trade-offs between different dimensions of the evaluation framework. When the model was trained on a single seed with 5000 iterations. Both novelty and diversity performed well throughout the training. The model was generating novel and diverse payloads, and novelty stayed above 0.8 during this time. However, the WAF bypass was zero, and none of the generated payloads evaded the WAF as shown in Figure 15.

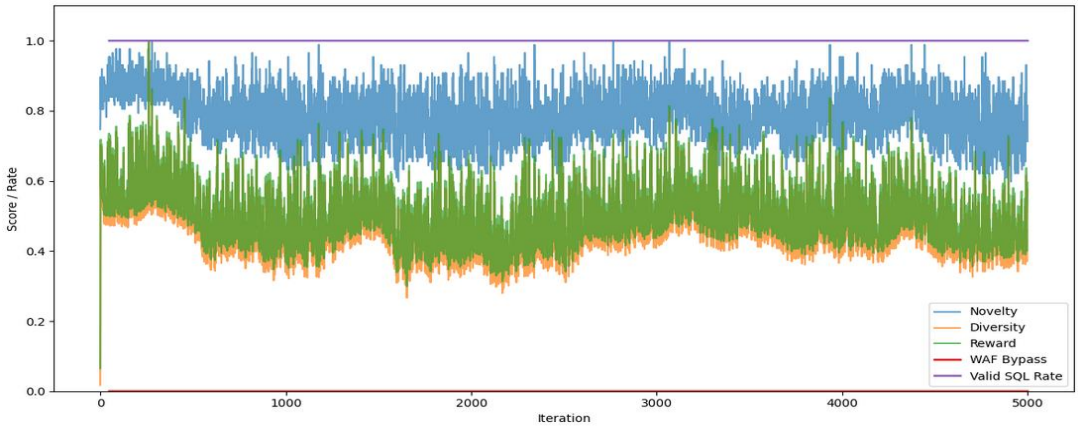


Figure 15. All dimensions at Single Seed Payload

The role of novelty and diversity was also observed by providing only a single seed payload. Since the single payload had less novelty and diversity. So, the model was unable to bypass WAF.

However, when the model was trained on a batch of payloads, Novelty was high, which shows the model generated novel payloads (0.85-0.95). Diversity fell after 2500 iterations and recovered. However, WAF bypass was observed shown in Figure 17 when the model was trained on a batch of seed payloads. Novelty and diversity in the seed payloads played a critical role in WAF evasion because model was able to generate more combinations based on given set of payloads. Rewards followed diversity, and it was observed that where diversity was high, WAF bypass was also spiked. In other words, novelty and diversity played a pivotal role in WAF bypass, as shown in Figure 16.

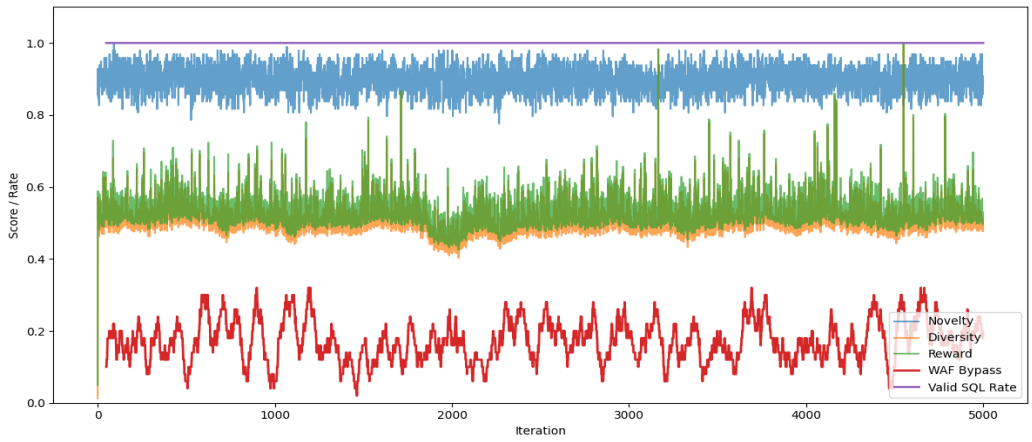


Figure 16. All dimensions at the Batch of Seed Payloads

The reward graph showed fluctuations and dips, as the reward was based on novelty, diversity, validity, semantic success, and WAF bypass. When these dimensions scored high, the reward was also high, indicating that the model was learning through exploration to optimize rewards based on a defined feedback loop. Therefore, the reward decreased when only a single payload was provided, due to a lack of novelty and diversity in the seed payload, as shown in Figure 17.

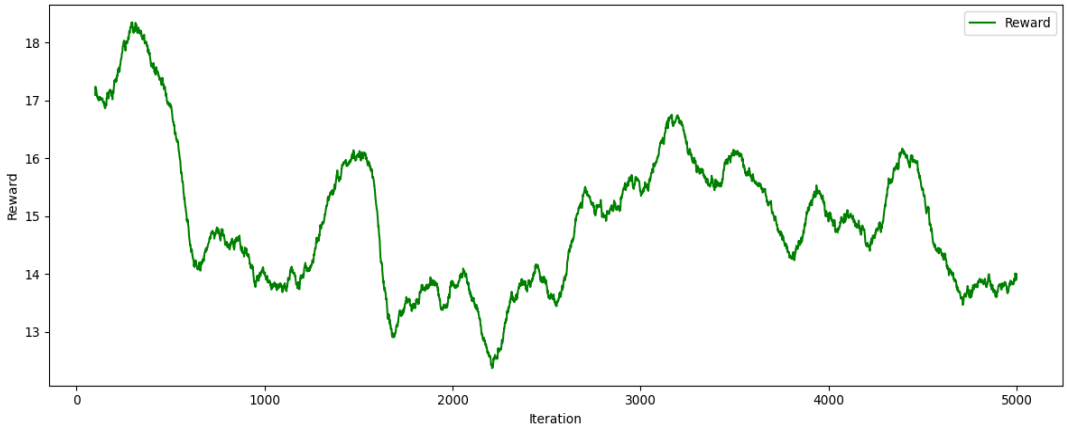


Figure 17. Reward graph at a Single Seed

However, when models were trained on a batch of seed payloads, the behavior of the model was stable as compared to when trained on a single seed payload. In addition, the overall reward increased as shown in Figure 18. The reason behind the increase in rewards was due to the increase in novelty and diversity in the seed payloads.

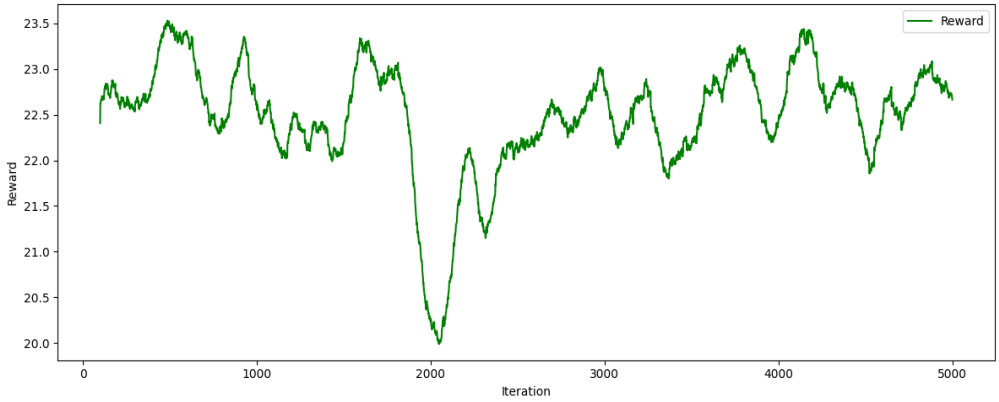


Figure 18. Reward Progression at Batch Payload

In addition, the reward increased when the model bypassed the firewall. This happens when the mutation increases diversity in payloads, and the model finds diverse payloads that can successfully bypass the WAF. Therefore, the overall reward increased. Similarly, the Novelty graph showed higher values on a batch of payloads, while lower values at the single seed, as can be seen in Figure 19.

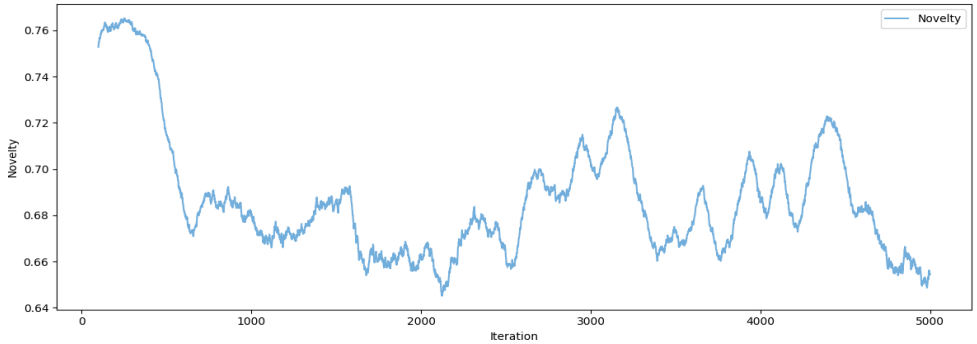


Figure 19. Novelty at Single Payload

Figure 19 shows that novelty was high at the beginning because the model was in exploration mode due to new and unexpected payloads. However, when the model learned the successful payload or the one that had increased the reward, then it started exploiting by tweaking around the effective payloads, and as a result, novelty decreased, which was a shift from exploration to exploitation mode. However, the overall value of novelty was low when a single seed payload was supplied, as observed in Figure 19, compared to a batch of payloads, as shown in Figure 20.

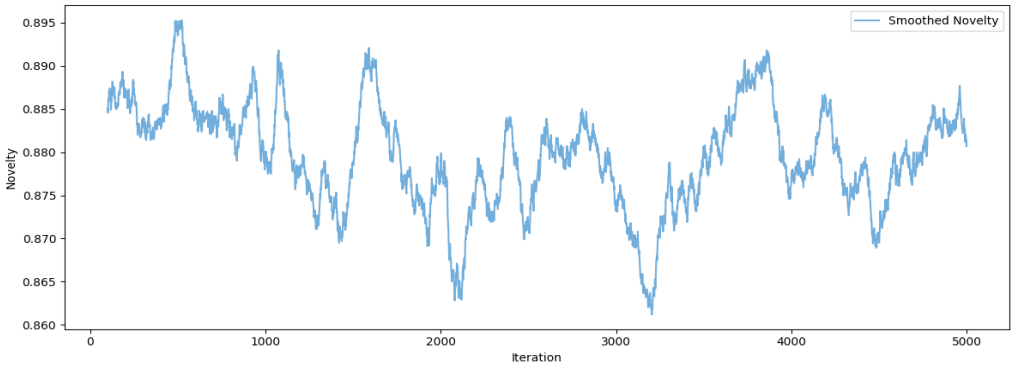


Figure 20. Novelty at Batch of Payloads

The novelty of the payload was highest at the first successful bypass, then the model narrows down its exploration to successful characters that bypassed the WAF. Only focusing on these effective characters shows similarity to a human pen tester, which narrows down its search to successful characters to come up with a refined attack payload. In 100 to 200 iterations, novelty was high because the model was exploring at this stage, then it settled down around 0.83–0.86, where the model refined its payload by keeping a balance between exploration and exploitation. Likewise, the diversity also increased on a batch of payloads and showed higher values. Figure 21 was the diversity graph at a single seed payload, where the maximum value was 34.

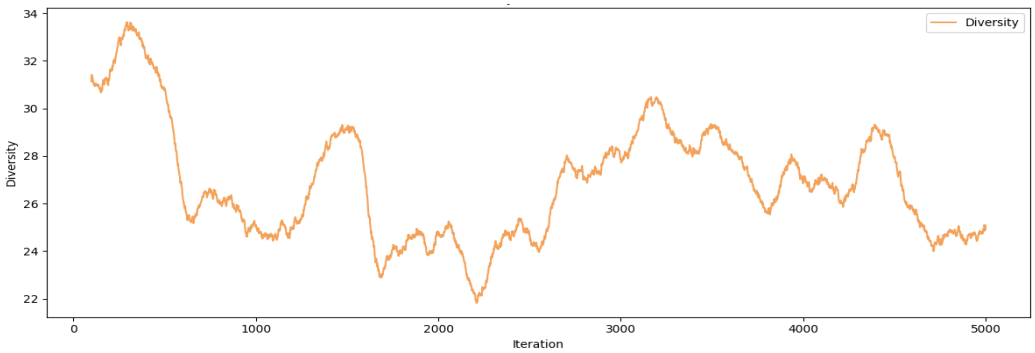


Figure 21. Diversity at a Single Seed Payload

On the other hand, the diversity of the batch payload was higher compared to a single seed, as shown in Figure 22. The oscillations shown were due to the model's exploration behavior.

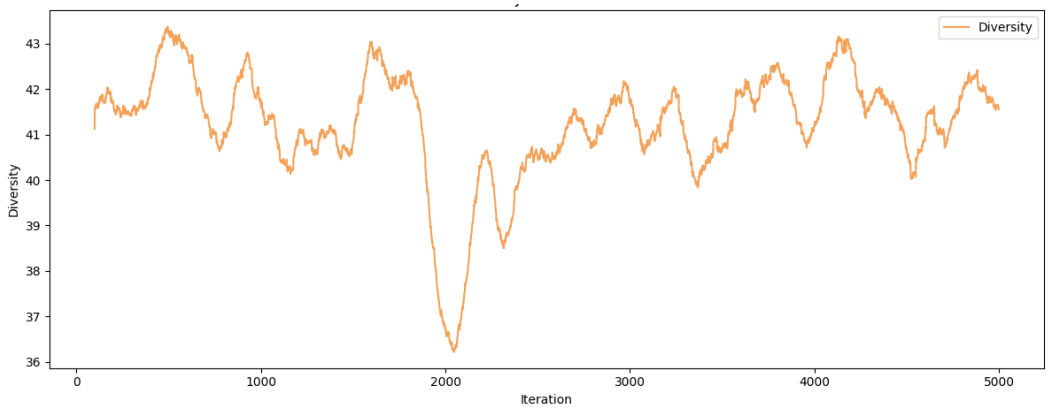


Figure 22. Diversity at a Batch of Payloads

Validity was constant throughout the training, as shown in Figure 23, and there was no change in whether the model was trained on a single payload or a batch of payloads. SQLparser ensured that at least one SQL keyword was in the generated payload.

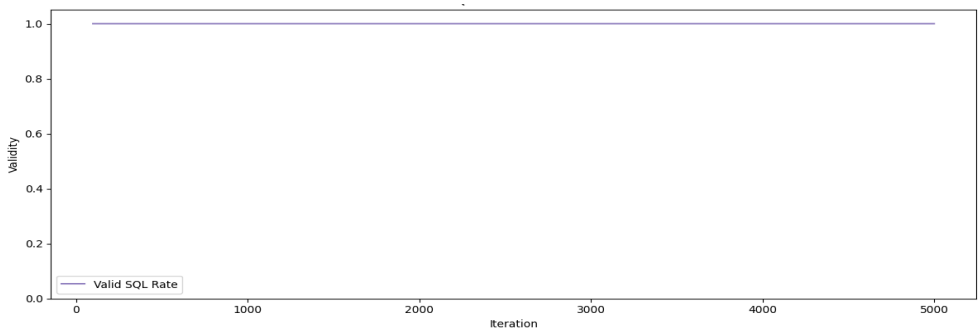


Figure 23. Validity of Generated Payloads at a Batch of Seed Payloads

The analysis of generated payloads showed that the model also preserved the syntactic validity by keeping at least one SQL keyword in generated payloads. The comparison of the generated payloads at a single seed and a batch of seed payloads is shown in Table 13.

Table 13. Payloads generated by a single seed and a batch of payloads

Generate Payloads by Single Seed	Generated Payloads by Batch of Payloads
'/**/or/**/'1'='1'/*-- */#u/qUx\/md,,(SU<Aby2GF/**/wi/**/	admin' || 1=1#j(ZK%%-;;-TS =IPf@pnL_A)9`
' oR '1'='1'/*-- */VP7wB|9r~_5Dd_3ocNsw'v62#6	admin'%09or%09'1'='1'/*EEEEEEE
' || '1' between '1'-- WHQoeCY!)#o"YCWE!_FXSq8Rf|	`)%09or%09(`x`)=(`x1EEE
\' or \'1\' between \'1\'-- 00rLfEr|sGz2e8)E#dKsKZ/5S	"%09or%09""%09"fff
'+or+'1'='1'/*-- */rw,~Fj3DAa~A2nzjk>VX*STgNt	")%09or%09("x")=("x

However, semantic success stayed constant at 0 as evidenced in Figure 24. The reason for zero was the inability of the model to directly integrate with a vulnerable web application. To generate semantically correct payloads, direct integration was required. For this research, we relied on Burp Suite to deploy payloads in testing using its Intruder functionality. In order to obtain semantically correct payloads, the model needs better guidance or context through direct integration with the application.

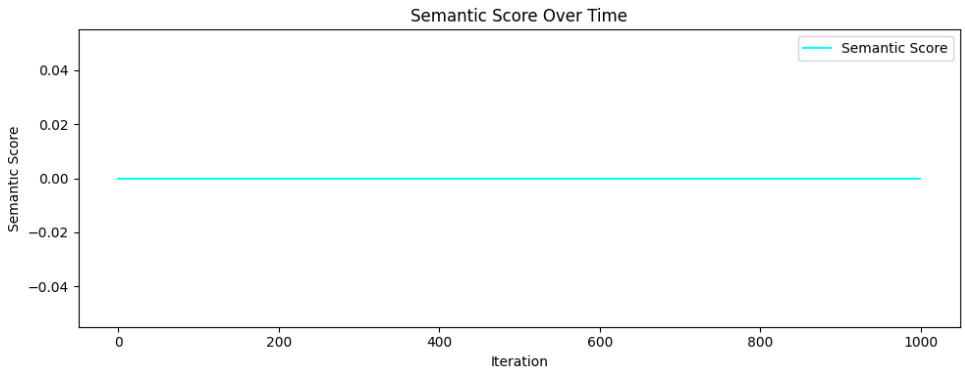


Figure 24. Semantic of Generated Payloads at a Single Seed Payload

As mentioned earlier, there was no bypass for a single payload, but multiple WAF bypasses when trained on a batch of payloads. The model found creative ways to evade WAF using combinations of multiple payloads. The flat red line at the bottom of Figure 25 was the indication that none of the mutated payload was able to bypass WAF when a single payload was supplied.

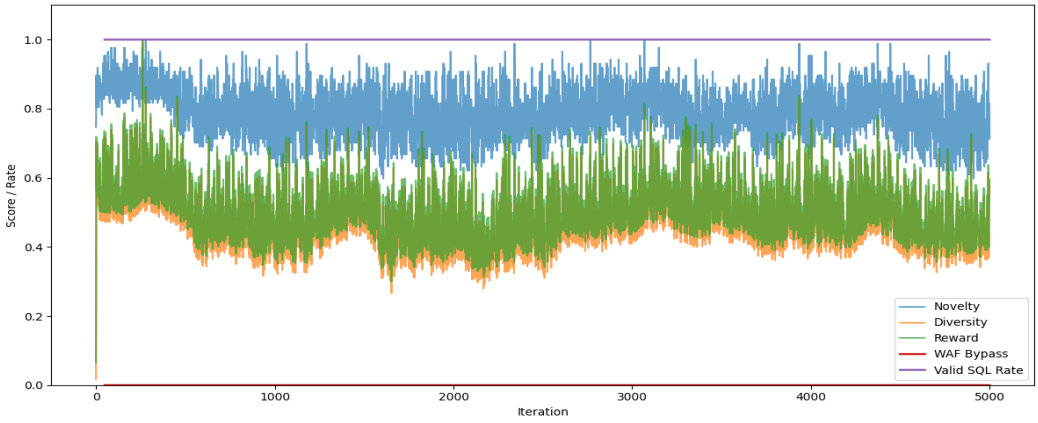


Figure 25. WAF Bypass at the Single Seed Payload

The red line in Figure 26 is the line showing periodic WAF bypass, and it happened when novelty and diversity were high due to a batch of payloads.

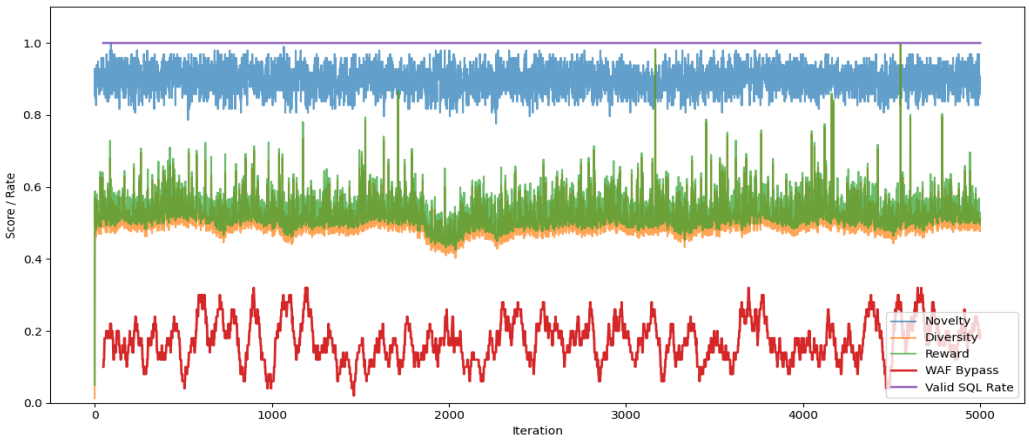


Figure 26. WAF Bypass at the Batch of Seed Payloads

The model was trained on 5000 iterations for a single seed and a batch of seed payloads. The results showed that the model was not even generating a WAF bypass payload, but at the same time, it mutated a single payload 5000 times, and it was still unable to evade the WAF. Table 14 showed that 5000 iterations on a single payload were not enough to evade the firewall. On the other hand, a batch of seed payload added diversity and novelty; as a result, the rate of WAF bypass increased.

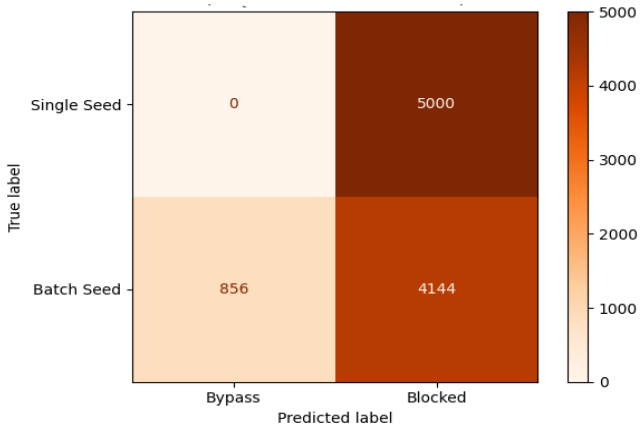


Figure 27. WAF Bypass Over Time at a Batch Payload

Figure 27 shows that by expanding the seed payloads, the model was able to learn better to find creative ways to evade WAF. In addition, when a model exploits WAF and gets high rewards, it starts exploring that specific bypass technique and creates payloads using the same techniques with different combinations.

Table 14. WAF Bypass Progress Over Time at a Single vs a Batch of Payloads

Iteration Range	Total Payloads	Single Seed WAF Bypass	Batch WAF Bypass
0-500	500	0	91
501-1000	1000	0	125
1001-1500	1500	0	282
1501-2000	2000	0	315
2001-2500	2500	0	385
5000	5000	0	856

It was observed that novelty and diversity were critical when it comes to bypassing WAF. When there was less novelty and diversity in the seed payload, there was less chance of bypassing WAF, as we had seen in the single seed case. In addition, the increase in iterations means models find more creative ways to bypass the firewall, and this increases with each iteration, as shown in Table 14. At 2500 iterations, the model discovered 385 ways to bypass the firewall, and the number increased to 856 at 5000 iterations. This data supported our hypothesis that ML models can discover complex and undiscovered vulnerabilities if we increase novelty, diversity of attack payloads. Table 15 shows the sample WAF payloads generated by the model.

Table 15. Payloads Generated by Model to Evade WAF

WAF Bypass Payloads
admin' || 1=1#j(ZK%%-;;-TS =IPf@pnL_A)9`
admin'%09or%09'1'='1'/*EEEEEEE
`)%09or%09(`x`)=(`x1EEE
"%09or%09""%09"fff
")%09or%09("x")=("x

In the analysis of WAF bypassed payloads, it was evident that when the model bypassed the WAF using an encoded mutation technique, it started exploring and exploiting the same payloads to gain high rewards by generating creative and novel payloads. As mentioned earlier, WAF bypass was high when the model was trained on a batch of seed payloads as compared to a single payload. This also highlights blind spots in WAF, where WAF was weak and unable to stop encoded SQLi attacks.

Further box plot analysis showed that novelty was higher, and diversity was tightly clustered for payloads that bypassed WAF, while it was widespread for the payloads that were blocked by WAF, as reflected in Figure 28. A narrow box plot demonstrated that the model found characters that bypass WAF and then started exploring around those characters using different variations. When the model found a sweet spot where it attained a higher reward for bypassing WAF, it started exploring in that range instead of random exploration of characters. This was evident in Table 15, where we noticed an encoding character pattern in bypassed payloads.

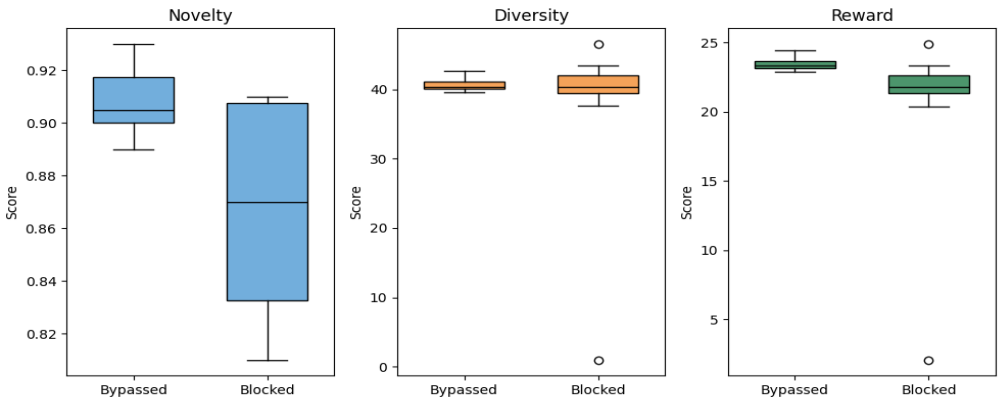


Figure 28. Comparison of Bypassed and Blocked Payloads

4.5 Comparison of RL LSTM with Text GAN, Stacked LSTM, and SQLmap

To evaluate the effectiveness of the reinforcement learning–based SQL injection payload generator, a comparative analysis was conducted against two alternative standalone models, TextGAN and stacked LSTM, that were developed for this research. Both models were standalone without any direct feedback from the environment, which limited their ability to consistently produce functional, evasive payloads. The TxtGAN and Stacked LSTM were trained on datasets of attack payloads to overcome the limitations of existing SQLi exploitation solutions. Both were used as strong generative baselines for comparison with an upgraded version of the model with feedback. On the other hand, a feedback-based model guided by an evaluation metric-based reward function, including SQL validity, WAF bypass success, novelty, and diversity, significantly outperformed both baselines in terms of bypass rate, structural validity, and adaptability. These results in Table 16 highlight that the lack of feedback could compromise the semantics of SQLi attack payloads, which emphasizes the importance of reinforcement feedback and goal-directed generation in adversarial security tasks.

In this study, the reinforcement learning–based payload generator was benchmarked against SQLmap and two baseline models developed in parallel: a supervised Stacked LSTM and a TextGAN-based generator. These baselines reflect conventional machine learning strategies for text generation using static datasets.

Table 16. Comparison of SQLi Tools and Models

Metric	*SQLmap*	*TextGAN*	*Stacked LSTM*	*RL LSTM with Feedback*
Syntactic Validity	100	100	100	100
Semantic Success (%)	100	0	0	0
Novelty	0.53	0.65	0.55	0.79
Diversity (%)	77.48	100	100	39.91
WAF Bypass (%)	0	0	0	17.12%

Note: *Novelty was measured by cosine distance (range: 0-1); a value of 1 indicates maximum dissimilarity from the reference payload.*

It was noticed that the RL-LSTM model with a feedback loop outperforms the standalone TextGAN and stacked LSTM models due to their reward mechanism. This evaluation metric directs the model to generate payloads in a confined state where the model was able to balance the novelty, diversity, validity, and WAF bypass. Figure 29 shows the comparison of all models evaluated based on a five-dimensional framework and provides insight into the missing capabilities of tools and models.

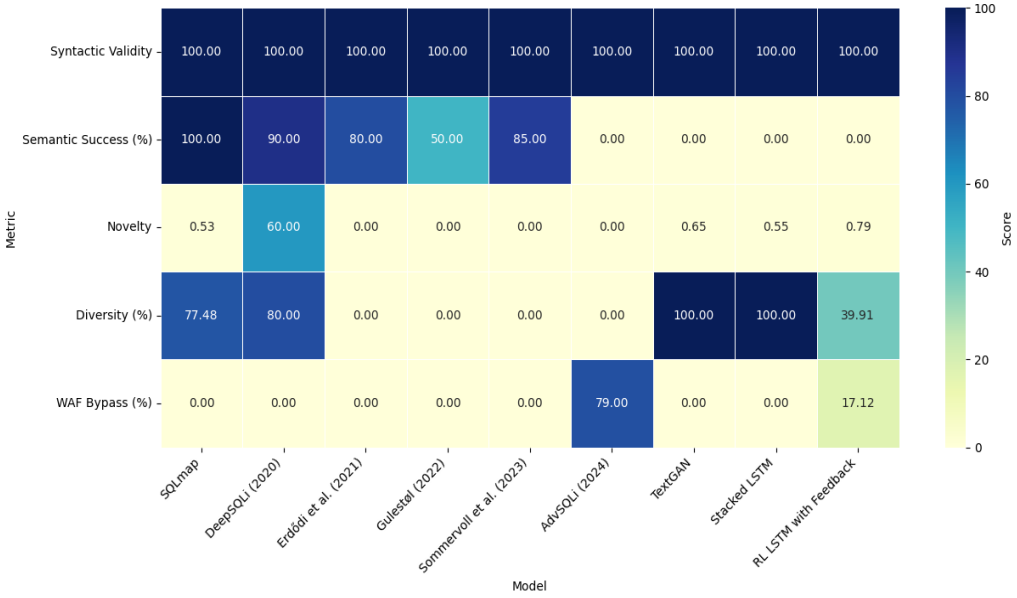


Figure 29. Model Comparison Based on Evaluation Metric

Chapter 5—Discussion and Conclusions

This chapter is devoted to the conclusion of the praxis and describes results in detail in the context of the research questions. The objective is to provide an overview of the research and contextualize the findings with research questions and hypotheses that guided this research on SQL injection (SQLi) payload generation and exploitation. Furthermore, this chapter highlights the contributions and limitations and provides recommendations for future work.

5.1 Discussion

5.1.1 RQ1 What are the limitations of existing solutions for exploiting complex SQL Injection vulnerabilities?

H1: *Existing SQL injection exploitation solutions have significant limitations in generating novel, diverse, valid, semantically correct, and WAF-evading attack payloads to exploit complex SQL injection vulnerabilities while having a balanced approach.*

The evaluation of existing SQLi exploitation showed that none of the solutions provides a balanced approach between novelty, diversity, validity, and WAF bypass. Some of them were heavily focused on WAF bypass, while others leaned towards novelty. There was a lack of proper balance between all these dimensions to ensure that the payloads are functional and novel.

5.1.2 RQ2 How can AI models improve the novelty, diversity, validity, and effectiveness of SQL injection attack payloads during exploitation while keeping a balance?

H2: *The sequence-based AI models such as TextGAN and stacked LSTM can increase the novelty, diversity, and validity in SQLi attack payloads while having a balance compared to those produced by existing SQLi solutions.*

The experimentation results based on our proposed methodology support our hypothesis that TextGAN and LSTM were able to improve the novelty, diversity, and validity of attack payloads when trained on the SQLi payloads dataset. We observed that both machine learning models performed better than SQLmap in terms of Novelty, diversity, and validity due to their generative capabilities. On the other hand, non-ML-based solutions were based on predefined syntax, which was their inherited weakness in generating novels, diverse attack payloads.

5.1.3 RQ3 Can reinforcement learning and stacked LSTM with an evaluation-feedback loop improve the validity and WAF evasion of ML-generated SQLi payloads?

H3 *A feedback reward function based on the evaluation metric through reinforcement learning in stacked LSTM will enhance the WAF bypass rate.*

It was observed that machine learning models enhanced novelty, diversity, and validity when trained on a dataset of valid SQL attack payloads, but they were unable to generate evasive payloads capable of bypassing WAF without any feedback mechanism. Therefore, the integration of an evaluation metric as feedback improved the WAF bypass due to the high reward. The model learned the weak spots in WAF through exploration and started exploiting them using their novelty and diversity dimensions.

5.1.4 RQ4 What is the impact of the input seed payloads on novelty, diversity, and WAF bypass performance in reward-based models?

H4 *The use of multiple input seed payloads can introduce novelty and diversity in input data and as a result the WAF bypass rate will increase compared to a single seed payload.*

The results of the study showed that the model performed better in WAF bypass when provided multiple seed payloads. Introduction of multiple seed payloads enhanced novelty and diversity, and as a result, the model was able to create novel, diverse, and evasive payloads using different combinations. Hence, the rate of WAF evasion increased. On the other hand, the single seed reduces the novelty and diversity in the attack payload, and as a result, WAF bypass was not achieved, even though the model generated 500 novel payloads.

5.2 Conclusions

The results of the study showed that existing SQLi exploitation lacks a balance between novelty, diversity, validity, semantic correctness, and WAF bypass. A perfect balance between dimensions creates a perfect balance for generating novel and functional attack payloads to discover complex SQLi vulnerabilities. The increase in novelty and diversity can enhance the rate of exploitation of SQLi vulnerabilities. The increase in novelty and diversity in a balanced way can be made possible through the integration of different machine learning algorithms. The integration of AI can improve the performance of existing solutions in terms of novelty and diversity by generating novel attack payloads. The capabilities of different AI models can create a sweet balance between these dimensions, such as GANs' generative capabilities, which can enhance novelty and diversity, while LSTMs can increase validity and semantic correctness of generated

payloads. Reinforcement learning models can force the model to ensure all these dimensions are maintained to create a balance.

5.3 Contributions to Body of Knowledge

This study offers several novel contributions to the field of SQLi generation and exploitation research:

- A comprehensive evaluation metric was proposed to evaluate and develop SQLi exploitation and attack generation solutions to analyze their strengths in terms of discovering novel and complex vulnerabilities.
- Two standalone models, TextGAN and stacked LSTM, without a feedback mechanism, were developed and trained on a dataset of known attack payloads to analyze their effectiveness based on an evaluation metric for advanced SQLi exploitation tools. Then, developed another hybrid RL-LSTM Model by embedding an Evaluation Metric as feedback to enforce the proposed dimension for successful SQLi attacks. This model can work on a single seed and a batch of payload seeds to enhance the novelty, diversity, and WAF bypass rate, which makes it practical for real-world applications where training data is limited or not available. This hybrid model was specifically developed for real-world use, and it provides flexibility in prioritizing specific dimensions, such as WAF bypass to find edge cases over others, or maintain balance.
- A relationship between novelty, diversity, and WAF bypass rate was measured and showed how both dimensions can enhance the rate of exploitation and play a critical role in discovering complex SQLi vulnerabilities.

5.4 Recommendations for Future Research

This research laid the foundation for the evaluation of SQLi exploitation and payload generation models based on an evaluation metric that can direct the process of improvement for such tools and solutions.

In the future, the research should focus on integrating the RL-LSTM evaluation embedded model with real web applications to enhance the usability and adaptability of the model for practical use during penetration testing, where it can handle sessions for direct interaction with the test environment.

Secondly, it is recommended to make this model a standalone tool for discovering novel and complex SQLi vulnerabilities that can assist teams in identifying unseen and novel SQLi payloads.

In addition, semantic correctness was still a challenge; therefore, future work should focus on improving the semantics of generated payloads by adding robust SQLi validation and semantic checks for implementation to enhance the performance of the model for practical use.

Lastly, the future work should explore expanding the mutation techniques based on real-world advanced techniques used by testers to enhance their effectiveness against advanced WAF for security testing.

References

Abdullah, H. S., Khalind, O. S., & Hamad, Z. O. (2023). *Analysis of SQLMAP Efficacy in Exploiting SQL Injection Vulnerabilities in Web Applications: A Case Study on DVWA*. 13–18. https://doi.org/10.1109/iceans58413.2023.10630454

Alanda, A., Satria, D., Mooduto, H. A., Dahlan, A. A., & Ardhana, M. I. (2021). Web Application Penetration Testing Using SQL Injection Attack. *JOIV : International Journal on Informatics Visualization*, *5*(3), 320. https://doi.org/10.30630/joiv.5.3.470

Althunayyan, M., Gope, P., Li, S., & Saxena, N. (2022). Evaluation of Black-Box Web Application Security Scanners in Detecting Injection Vulnerabilities. *Electronics*, *11*(13), 2049. https://doi.org/10.3390/electronics11132049

Baskaran, S., Susmitha Nrk, Sruthi S, & Thirulogaveni J. (2024, November 28). *A Comparative Analysis of Vulnerability Management Tools: Evaluating Nessus, Acunetix, and Nikto for Risk- Based Security Solutions*. https://doi.org/10.13140/RG.2.2.30040.71687

Becker, N., Reti, D., Ntagiou, E., Wallum, M., & Schotten, H. (2024). *Evaluation of Reinforcement Learning for Autonomous Penetration Testing using A3C, Q-learning and DQN*. https://doi.org/10.48550/arxiv.2407.15656

Bouafia, R., Amine, A., & Benbrahim, H. (2023). *Automatic Protection of Web Applications Against SQL Injections: An Approach Based On Acunetix, Burp Suite and SQLMAP*. *7*, 1–6. https://doi.org/10.1109/icoa58279.2023.10308827

Borgeaud, A. (2024, February 20). Global web application critical vulnerability taxonomy 2023. Statista. https://www.statista.com/statistics/806081/worldwide-application-vulnerability-taxonomy/

Boekweg, K. I. (2024). Developing a SQL Injection Exploitation Tool with Natural Language Generation. *Brigham Young University ScholarsArchive*. Retrieved from https://scholarsarchive.byu.edu/etd/10384.

Casola, V., Benedictis, A. D., Rak, M., & Villano, U. (2020). A methodology for automated penetration testing of cloud applications. International Journal of Grid and Utility Computing, 11(2), 267. https://doi.org/10.1504/ijguc.2020.105541

Chowdhary, A., Zhao, M., & Jha, K. (2023). Generative Adversarial Network (GAN)-Based Autonomous Penetration Testing for Web Applications. Sensors, 23(18), 8014. https://doi.org/10.3390/s23188014

Derbyshire, J. (2022, June 22). *Why organizations can no longer rely on traditional security solutions to protect their data*. BCS. Retrieved from https://www.bcs.org/articles-opinion-and-research/why-organisations-can-no-longer-rely-on-traditional-security-solutions-to-protect-their-data/

Dunmore, A., Kwak, J., Sabrina, F., & Jang-Jaccard, J. (2023). A Comprehensive Survey of Generative Adversarial Networks (GANs) in Cybersecurity Intrusion Detection. *IEEE Access*, *11*, 76071–76094. https://doi.org/10.1109/access.2023.3296707

Elder, S., Menzies, T., Shu, R., Metro, M., Zahan, N., Kozarev, V., & Williams, L. (2022). Do I really need all this work to find vulnerabilities? An empirical case

study comparing vulnerability detection techniques on a Java application. cornell university. https://doi.org/10.48550/arxiv.2208.01595

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. https://doi.org/10.1016/j.jisa.2021.102903

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative Adversarial Nets*. https://arxiv.org/pdf/1406.2661

Hajar, S., Abdul Rahim, F., & Jaafar, A. G. (2024). A Review of Penetration Testing Process For Sql Injection Attack. Open International Journal of Informatics, 12(1), 221–236. https://doi.org/10.11113/oiji2023.11n2.256

Harb, H., Eleyan, D., & Eleyan, A. (2021). SQL Injection Detection Tools Advantages and Drawbacks. International Journal of Wireless and Microwave Technologies, 11(3), 16–21. https://doi.org/10.5815/ijwmt.2021.03.03

Han, Y., Choi, Y., Lee, J., & Bae, J.-H. (2025). Feature Fusion Model Using Transfer Learning and Bidirectional Attention Mechanism for Plant Pipeline Leak Detection. *Applied Sciences*, *15*(2), 490–490. https://doi.org/10.3390/app15020490

Hilario, E., Azam, S., Imran Mohammed, K., Sundaram, J., & Shanmugam, B. (2024). Generative AI for pentesting: the good, the bad, the ugly. *International Journal of Information Security*, *23*(3), 2075–2097. https://doi.org/10.1007/s10207-024-00835-x

Huang, X., Han, M., & Deng, Y. (2024). A Hybrid GAN-Inception Deep Learning Approach for Enhanced Coordinate-Based Acoustic Emission Source Localization. *Applied Sciences*, *14*(19), 8811–8811. https://doi.org/10.3390/app14198811

Jagamogan, R. S., Abas, H., Ismail, S. A., & Hassan, N. H. (2022). Penetration Testing Procedure using Machine Learning. 58–63. https://doi.org/10.1109/icssa54161.2022.9870951

Joshi, P. N., Ravishankar, N., Raju, M. B., & Ravi, N. C. (2022). Surgical striking SQL injection attacks using LSTM. *Indian Journal of Computer Science and Engineering, 13*(1), 208-220. https://doi.org/10.21817/indjcse/2022/v13i1/221301182

Liu, M., Chen, T., & Li, K. (2020, July 18). *DeepSQLi: deep semantic learning for testing SQL injection*. https://doi.org/10.1145/3395363.3397375

Liu, Y., Xu, L., Yang, S., Zhao, D., & Li, X. (2024). Adversarial sample attacks and defenses based on LSTM-ED in industrial control systems. *Computers & Security*, *140*, 103750. https://doi.org/10.1016/j.cose.2024.103750

Liu, Y., & Dai, Y. (2024). Deep Learning in Cybersecurity: A Hybrid BERT–LSTM Network for SQL Injection Attack Detection. *IET Information Security*, *2024*, 1–16. https://doi.org/10.1049/2024/5565950

Lin, J., & Dong, W. (2024). SPAFuzz: Web security fuzz testing tool based on fuzz testing and genetic algorithm. *Proceedings of the 4th International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, 190–192. IEEE. https://doi.org/10.1109/ICCECE61317.2024.10504208

Lu, D., Liu, L., Fei, J., & Li, Z. (2022). *A GAN-based Method for Generating SQL Injection Attack Samples*. 1827–1833. https://doi.org/10.1109/itaic54216.2022.9836726

Mamilla, Reddy, S. (2021). *A Study of Penetration Testing Processes and Tools A Study of Penetration Testing Processes and Tools*. https://scholarworks.lib.csusb.edu/cgi/viewcontent.cgi?article=2394&context=etd

Masafumi, I. Takaesu, T. Terada, T. Kudo, and T. Yoneyama, "GyoiThon - Next generation penetration test tool," Black Hat Asia 2018 Arsenal, pp. 2–3, 2018.

Mirza, M., & Osindero, S. (2014). *Conditional Generative Adversarial Nets*. https://arxiv.org/pdf/1411.1784

Nair, S. S. (2024). Securing Against Advanced Cyber Threats: A Comprehensive Guide to Phishing, XSS, and SQL Injection Defense. Journal of Computer Science and Technology Studies, 6 (1), 76–93. https://doi.org/10.32996/jcsts.2024.6.1.9

Nguyen-Duc, A., Do, M., Hong, Q., & Khac, K. (2021). On the combination of static analysis for software security assessment -- a case study of an open-source e-government project. Cornell University. https://doi.org/10.48550/arxiv.2103.08010

Ojagbule, O., Haddad, R. J., & Wimmer, H. (2018). *Vulnerability Analysis of Content Management Systems to SQL Injection Using SQLMAP*. 1–7. https://doi.org/10.1109/secon.2018.847913

OWASP. (2024). SQL Injection. OWASP. https://owasp.org/www-community/attacks/SQL_Injection

PTES. (2014, August 16). *The Penetration Testing Execution Standard*. Pentest-Standard.org. http://www.pentest-standard.org/index.php/Main_Page

Qu, Z., Ling, X., Wang, T., Chen, X., Ji, S., & Wu, C. (2024). AdvSQLi: Generating adversarial SQL injections against real-world WAF-as-a-service. *IEEE Transactions on Information Forensics and Security, 19*, 2623–2636. https://doi.org/10.1109/TIFS.2024.3350911

Rahman, M. A., Siddik, M. S., Ahmed, B., & Amjad, M. (2020). *Analyzing Web Application Vulnerabilities*. 1–6. https://doi.org/10.1145/3377049.3377107

Rafiq, H., Aslam, N., Issac, B., & Randhawa, R. H. (2022, May 2). *An Investigation on Fragility of Machine Learning Classifiers in Android Malware Detection*. https://doi.org/10.1109/infocomwkshps54753.2022.9798161

Rane, N., & Qureshi, A. (2024). *Comparative Analysis of Automated Scanning and Manual Penetration Testing for Enhanced Cybersecurity*. https://doi.org/10.1109/isdfs60797.2024.10527240

Renjith, G., Laudanna, S., Aji, S., Visaggio, C. A., & Vinod, P. (2021). GANG-MAM: GAN based enGine for Modifying Android Malware. *arXiv preprint arXiv:2109.13297*. https://doi.org/10.48550/arXiv.2109.13297

Rostovcev, N. (2024, February 6). ResumeLooters: A look into a credential-stealing group targeting job seekers. Group-IB. https://www.group-ib.com/blog/resumelooters/https://www.group-ib.com

Rose, J. D., Tyagi, A., Tushar Rathee, & Chopra, K. (2022). Web Application Vulnerability Exploiter - WAVE.*International Journal of Engineering Research & Technology*, *10*(13). https://doi.org/10.17577/IJERTCONV10IS13003

Singh, N., Meherhomji, V., & Chandavarkar, B. R. (2020). *Automated versus Manual Approach of Web Application Penetration Testing*. 1–6. https://doi.org/10.1109/icccnt49239.2020.9225385

Sommervoll, Å. Å., Erdődi, L., & Zennaro, F. M. (2023). Simulating all archetypes of SQL injection vulnerability exploitation using reinforcement learning agents. *International Journal of Information Security*, *23*(1), 225–246. https://doi.org/10.1007/s10207-023-00738-3

Stefinko, Y., Piskozub, A., & Banakh, R. (2016, February 1). *Manual and automated penetration testing. Benefits and drawbacks. Modern tendency*. https://doi.org/10.1109/tcset.2016.7452095

Thaqi, R., Vishi, K., & Rexha, B. (2022). Enhancing Burp Suite with Machine Learning Extension for Vulnerability Assessment of Web Applications. *Journal of Applied Security Research*, *18*(4), 1–19. https://doi.org/10.1080/19361610.2022.2096387

Tidy, J. (2023, June 6). MoveIt hack: What action can data-breach victims take? BBC News. https://www.bbc.com/news/technology-65820603

Van Houdt, G., Nápoles, G., & Mosquera, C. (2020). A review on the long short-term memory model. *Artificial Intelligence Review*, *53*(8), 5929–5955. https://doi.org/10.1007/s10462-020-09838-1

Vijayan, J. (2023, June 5). Mass exploitation of 0-day bug in MOVEit file transfer underway. Dark Reading. https://www.darkreading.com/application-security/mass-exploitation-0-day-bug-imoveit-file-transfer-underway

Yuan, Y., Lu, Y., Zhu, K., Huang, H., Chen, Y., & Zhang, Y. (2024). sqlFuzz: Directed fuzzing for SQL injection vulnerability. *Electronics, 13*(2946). https://doi.org/10.3390/electronics13152946

Zhang, W., Mi, Y., Li, X., Li, Y., Shao, M., Zhi, G., & Zhang, H. (2022). Deep Neural Network-Based SQL Injection Detection Method. Security and Communication Networks, 2022, 1–9. https://doi.org/10.1155/2022/4836289

Zhang, H., Jiang, L., & Dong, W. (2022). ZokFuzz: Detection of web vulnerabilities via fuzzing. *Proceedings of the 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, 281–285. IEEE. https://doi.org/10.1109/ICCECE54139.2022.9712748

Zhao, D. (2020). Fuzzing Technique in Web Applications and Beyond. *Journal of Physics: Conference Series*, *1678*(1), 012109. https://doi.org/10.1088/1742-6596/1678/1/012109