

Article

A Secure Prescription System with Machine Learning for SQL Injection Detection

Savina Mariettou^{1*}, Constantinos Koutsojannis², Vassilis Triantafyllou³

¹ Electrical and Computer Engineering Department, University of Peloponnese, Patras, Greece, s.mariettou@go.uop.gr

² Professor of Medical Physics & Electrophysiology, Director of Health Physics & Computational Intelligence Laboratory, Physiotherapy Department, School of Health Rehabilitation Sciences, University of Patras, Patras, Greece, ckoutsog@upatras.gr

³ Professor of Network Technologies and Digital Transformation lab, Electrical and Computer Engineering Dpt., University of Peloponnese, Patras, Greece, vtriantaf@uop.gr

*Corresponding author: s.mariettou@go.uop.gr

Received: 29 April 2025; **Revised:** 23 June 2025; **Accepted:** 30 June 2025

Abstract: This research introduces a secure, web-based prescription system designed to monitor antibiotic consumption and reduce the misuse of critical antibiotics in clinical environments. The system's user interface supports structured documentation and justification of antibiotic use, serving as a clinical surveillance tool that promotes responsible prescribing and contributes to the prevention of hospital-acquired infections through improved antimicrobial stewardship. To ensure robust data protection, the system was evaluated under simulated cyberattacks, including unauthorized access, Denial-of-Service (DoS), Distributed Denial-of-Service (DDoS), and SQL injection attacks. In addition to standard security mechanisms such as Transport Layer Security (TLS) and Elliptic Curve Cryptography (ECC), the system integrates a machine learning-based module implemented in Python to enhance real-time SQL injection detection. The module leverages supervised learning algorithms to classify database queries as malicious or safe, enabling proactive defense against threats targeting sensitive medical records. By embedding machine learning into a secure clinical workflow, the system supports sustainable antibiotic management in hospitals, laying a foundation for scalable, intelligent, and secure e-health infrastructures.

Keywords: Healthcare Security, Data Protection, Healthcare Cybersecurity, Antibiotic Stewardship, Secure Data Management, Simulated Cyber Attacks

1. Introduction

The healthcare sector is at the forefront of data digitization, with technologies such as electronic health records, cloud platforms, and e-prescribing being leveraged to store, process, and analyze data [1]. This system enhances efficiency and improves healthcare service quality, contributing to a more reliable and effective system of care [2]. However, the rise of antimicrobial resistance has emerged as one of the ten most critical global public health threats, causing 5 million deaths annually, of which more than half a million occur in Europe and Central Asia [3]. Antimicrobial resistance arises when microorganisms resist the drugs designed to combat them, rendering infections increasingly difficult to treat. Misuse and overuse of antibiotics, particularly in cases of unnecessary prescriptions or non-prescription usage, exacerbate the spread of these "superbugs". If left unchecked, AMR could account for up to 390.000 deaths annually by 2050 [4].

Despite advances in healthcare technology, challenges persist in safely integrating health services. Many hospitals still lack integrated information systems, resulting in the entire process, from diagnosis to treatment,

Copyright ©2025 Savina Mariettou, et al.

DOI: <https://doi.org/10.37256/cnc.3220257145>

This is an open-access article distributed under a CC BY license
(Creative Commons Attribution 4.0 International License)

<https://creativecommons.org/licenses/by/4.0/>

being recorded on paper rather than digitally [5]. This can be confusing, particularly in vulnerable populations, such as elderly people with chronic conditions, for whom safe and effective sharing of medical data during treatment remains a major concern [6]. In parallel, the rational use of antibiotics is a pressing issue, with studies showing excessive misuse in regions where antibiotics are readily available without prescriptions or where health systems lack strict monitoring protocols [7]. Addressing these dual challenges requires a multifaceted approach that combines robust system security with education and strategies for safe antibiotic use [8].

The proposed simulation-based information technology system offers a unified and secure platform to monitor and optimize antibiotic use, meeting these needs, ensuring transparency in antibiotic consumption, implementing strict data protection measures, and covering the process from diagnosis to treatment. It can serve as a foundation for creating national policies for antibiotics monitoring and infectious disease management worldwide, as many hospitals opt for paper recording and traditional storage for safety reasons.

System security and data integrity have been persistent challenges since 2010 [2]. However, practical solutions, particularly in Greek Health Care Institutions, will not be fully implemented until 2024 [5]. Encryption techniques once deemed reliable are now increasingly vulnerable to cyber threats, highlighting the urgent need for more resilient security frameworks in healthcare [2]. Additionally, proper medication use remains a critical concern, especially in systems where health professionals lack access to unified medical records, making diagnoses and treatments more complex [6].

Antibiotics are essential in treating bacterial infections, as they kill or inhibit their growth. However, improper use, such as unnecessary prescriptions or prolonged durations, has led to the alarming rise of antibiotic resistance. Implementing a simulation-based application system can support healthcare professionals in prescribing antibiotics appropriately, ensuring safer use and reducing resistance [9].

Literature has extensively explored machine learning-based SQL injection detection in general-purpose web applications. One study compares multiple classification algorithms such as SVM, Decision Tree, and Naive Bayes [10], while another employs deep learning models like CNN and MLP, combined with NLP techniques, to enhance detection performance [11]. However, these approaches remain limited to generic web platforms and do not address domain-specific needs in healthcare. A recent review on cybersecurity in digital health highlights that machine learning-based intrusion detection, particularly against SQL injection, remains underexplored in clinical systems, especially in modules such as electronic prescriptions [12]. This work addresses that gap through the integration of machine learning directly into the prescription workflow, enabling real-time detection of SQL injection attempts and enhancing database-layer protection in clinical environments where data integrity and patient safety are paramount. This integration represents the core novelty of the system, distinguishing it from existing approaches that focus solely on generic web applications.

This article introduces a secure electronic prescription system designed specifically for clinical environments, integrating machine learning to detect SQL injection attempts in real time. The core innovation lies in embedding security mechanisms directly within the prescription workflow, thereby enhancing database-level protection in contexts where data integrity and patient safety are critical.

Beyond security, the system also supports clinical goals by tracking antibiotic consumption and promoting their rational use. Its design facilitates better diagnosis and patient monitoring while maintaining high standards of data protection. By combining advanced cryptographic techniques with intelligent threat detection, the system addresses key vulnerabilities in healthcare software and demonstrates the growing importance of cybersecurity in Medical Informatics. In Section 2, we present the design and functionality of the proposed system. Section 3 presents the server infrastructure, the system's security measures, and the enhancements we implemented to improve resilience with novel security features. Section 4 evaluates the system's resilience against cybersecurity threats by testing it under four simulated attack scenarios: unauthorized access attempts, Denial-of-Service (DoS) attacks, Distributed Denial-of-Service (DDoS) attacks, and Structured Query Language (SQL) injection attempts. These tests assess the system's ability to withstand real-world security challenges. Section 5 focuses specifically on SQL injection attacks, proposing an enhancement through machine learning to improve detection and system efficiency. Section 6 concludes with a summary of our findings, and Section 7 outlines potential directions for future work.

2. Prescription System Design and Functionality

Infrastructure encompasses the physical structures, facilities, and organizational systems that form the foundation of a nation's economy, health, and security [13]. In healthcare, the adoption of electronic clinical prescription systems has opened opportunities for more effective and timely monitoring of medication management, as seen in aged care initiatives. For instance, the National Aged Care Medication Roundtable demonstrated how leveraging medication administration datasets can enhance safety and decision-making in real-

world contexts [14]. Similarly, programs like the Agency for Healthcare Research and Quality (AHRQ) Safety Program for Improving Antibiotic Use underscore the value of structured approaches in utilizing clinical data and decision-making frameworks to optimize antibiotic stewardship while addressing gaps in implementation across diverse healthcare settings [15].

Building on these insights, our system tackles key challenges such as inappropriate antibiotic use and the absence of real-time monitoring. It shifts the focus from theoretical approaches to practical, actionable solutions that enhance prescribing accuracy and patient safety. The system is based on a client-server architecture, where infectiologists and physicians (clients) interact with a web-based interface, as a central server handles processing and communication with the database. The system integrates modern web technologies, including HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) for user interface design and HyperText Preprocessor (PHP) for database connectivity, to create a scalable and secure infrastructure. The development process was structured into four distinct stages, as detailed below:

Stage 1: Form Design and Collaboration

In the initial phase, we focused on designing the fill-in form in collaboration with the surveillance team on antibiotics. The form facilitates the systematic registration of patient information, capturing their feedback, treatment progress, and responses to prescribed medications. What sets this system apart from a simple data collection form is that it includes an additional verification process by the attending physician and approval by a virulence specialist at a two-week interval.

An important feature of the system is the approval workflow for ongoing treatment. Specifically, if the patient requires continued therapy, approval must be granted within 14 days [16]. Otherwise, the treatment should be discontinued, and the patient's infection progress must be reassessed. Furthermore, the expert physician can recommend or restrict specific antibiotics based on clinical data and usage guidelines. Notably, research has shown that the duration of antibiotic treatment can be reduced if the patient shows clinical improvement. Specifically, community-acquired pneumonia (CAP) can be treated in 3 to 5 days, acute exacerbation of chronic obstructive pulmonary disease (AECOPD) in 3 to 6 days, hospital-acquired pneumonia (HAP) in 7 days, and streptococcal pharyngotonsillitis in 4 to 5 days. However, these recommendations are still being studied, and further investigations are needed to confirm their effectiveness [17].

Stage 2: Database Integration

To illustrate the implemented system logic, an Activity Diagram (Figure 2) is provided, describing the flow from user input through PHP processing to database storage. This abstraction reflects the system's functional structure and data handling pipeline.

Stage 3: Server Setup and Accessibility Expansion

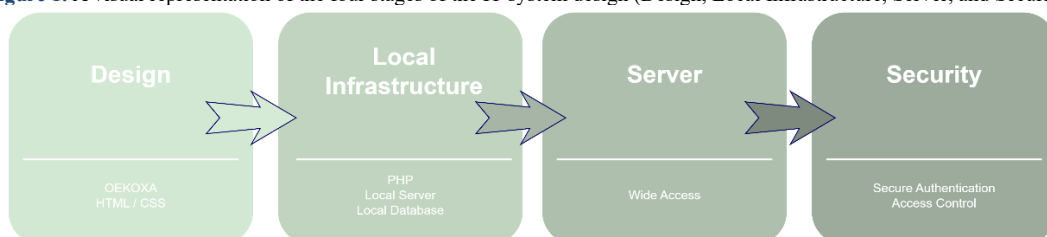
We built a dedicated server in the third stage to expand system accessibility. This development expanded system accessibility while identifying and mitigating potential security risks and threats to the system infrastructure.

Stage 4: Security Mechanism Implementation

The final and most critical stage involved implementing a specialized security mechanism. This included password protection for the surveillance team tasked with ensuring the proper and responsible use of antibiotics. This security mechanism safeguards the integrity of the prescription system and restricts access to authorized personnel only.

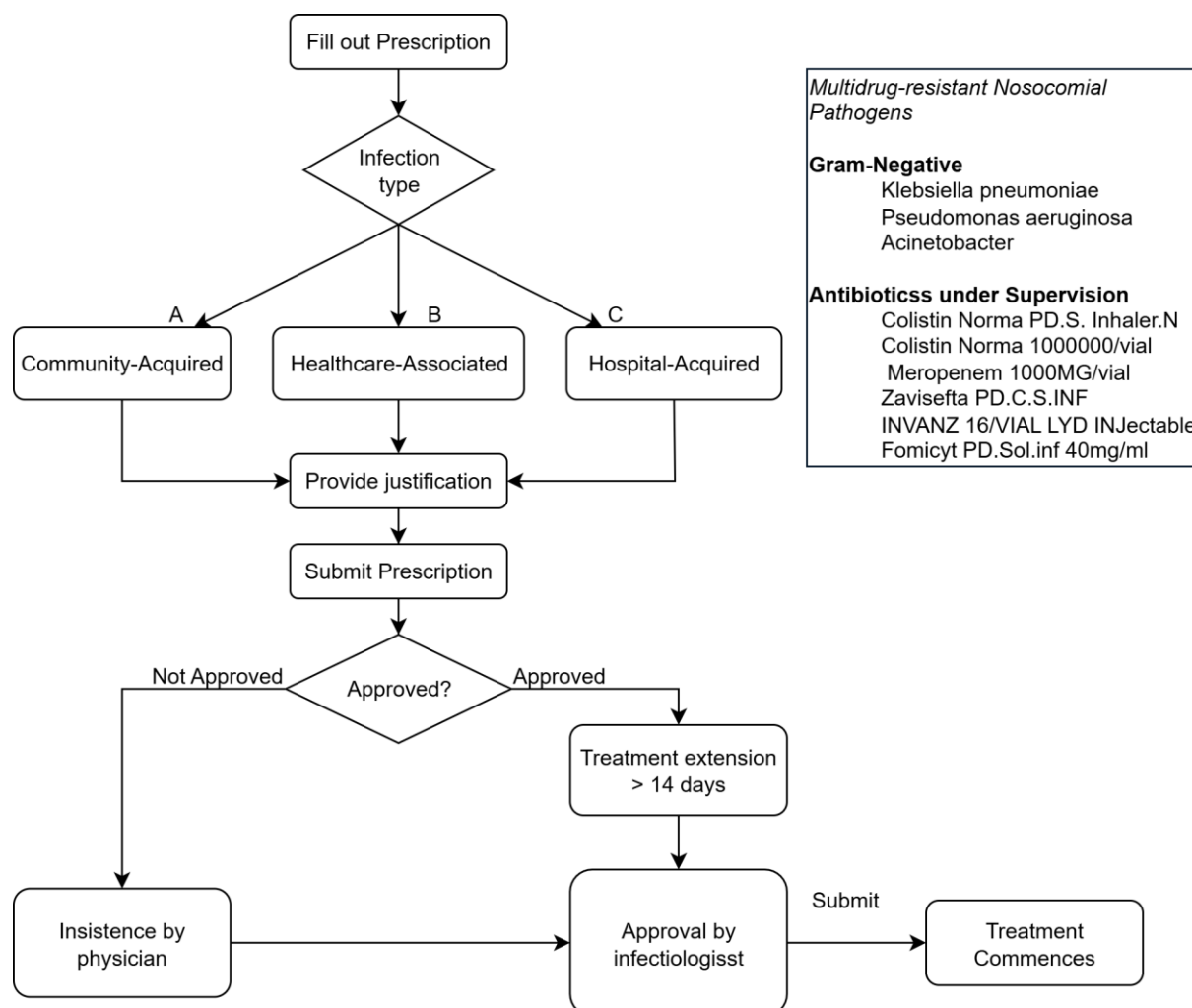
After completing the design and implementation phases, the information technology system was systematically structured into four distinct stages, as illustrated in Figure 1. These stages include the initial design, local infrastructure development, the transition to remote hosting for broader accessibility, and the robust security system deployment.

Figure 1. A visual representation of the four stages of the IT system design (Design, Local Infrastructure, Server, and Security).



The fill-in prescription form design incorporated several mandatory fields to ensure accurate and complete data registration. These fields included identifying the hospital, the initial diagnosis date, and, where applicable, approval details from an infectiologist, as illustrated in Figure 2.

Figure 2. Activity Diagram for the Justification and Approval Process of “Protected” Antibiotics



Certain antibiotics incorporated into the prescription workflow—such as Colistin, Meropenem, and Ceftazidime–Avibactam (Zavicefta)—are considered last-line agents for treating multidrug-resistant (MDR) Gram-negative pathogens, including *Klebsiella pneumoniae* and *Pseudomonas aeruginosa*. International health authorities widely recognize these pathogens as critical threats in hospital-acquired infections. Due to the limited therapeutic alternatives available, these agents are subject to strict surveillance and control. Recent studies have highlighted the potential of synergistic combinations involving these drugs to restore efficacy against resistant strains [18]. However, such effectiveness is highly dependent on timely and accurate administration. Therefore, any compromise in the prescription system, such as unauthorized access or a successful SQL injection, could delay treatment or lead to prescription errors, directly affecting patient safety. This further emphasizes the need for robust, database-level security in systems handling critical clinical information.

3. Server and Security

Following the analysis of 32 healthcare safety systems, a critical need emerged to enhance the protection of medical data, alongside a noticeable absence of experimental testing to validate the robustness of many existing security mechanisms [19]. To address these concerns, we developed and tested a custom system for monitoring antibiotic usage, incorporating additional security measures to enhance resilience. This process followed a structured methodology, detailed in the subsections below.

Initially, the system was tested in a local environment using XAMPP, before being deployed online via VistaPanel. The deployment began with an RSA 2048-bit Secure Sockets Layer (SSL) certificate, which was later replaced with an Elliptic Curve Cryptography (ECC)-based certificate to improve both security and performance.

ECC was selected as a more efficient and secure alternative to RSA. Offering equivalent cryptographic strength with significantly smaller key sizes, ECC reduces computational overhead and enhances performance [20]. It requires only about 10% of the bandwidth and storage space compared to RSA, making it highly suitable for resource-constrained applications such as our antibiotic tracking system, which incorporates embedded medical images [21]. The compact and robust mathematical properties of elliptic curves make ECC especially effective for secure communication in embedded environments.

To integrate ECC into the system, an ECC private key and Certificate Signing Request (CSR) were generated using OpenSSL. The CSR was submitted to a Certificate Authority (CA), and domain ownership was verified through a CNAME record. Once issued, the ECC SSL certificate was installed via VistaPanel. This setup utilized TLS 1.2, which supports ECC-based encryption and SHA-256 for message integrity and security [22]. The successful transition from RSA to ECC was validated using Qualys SSL Labs, which confirmed improved encryption speed, lower resource consumption, and overall stronger security.

In terms of database protection, the system uses secure communication protocols, including TLSv1.2 and TLSv1.3, and employs OpenSSL 1.1.1k (FIPS) for encryption. Data is managed through MariaDB 10.6.19 with the InnoDB 10.6.19 storage engine, which ensures transactional support and data consistency.

Beyond transport layer encryption, application-layer security was also implemented through machine learning. Specifically, a classification-based SQL injection detection system was integrated to monitor and flag potentially malicious queries in real time. This model introduces a proactive layer of defense by recognizing suspicious query patterns, complementing the traditional cryptographic measures. Further technical details on the machine learning integration are discussed in Section 5.

Before introducing machine learning, we conducted a series of simulated attacks to evaluate the system's resilience under standard configurations. The results of these penetration tests are presented in the following section.

4. Threats and Vulnerabilities: Security Testing through Simulated Attacks

The prescription system was tested against four types of attacks. We theoretically had access to a hospital's closed network in all the attacks. The attacks tested on my system include unauthorized access (Brute Force Attack), Denial of Service (DoS) attacks, Distributed Denial of Service (DDoS) attacks, and SQL Injection attempts. Below, we will explore each attack in detail, analyzing its methods, potential impact, and the results observed during the tests.

As part of the testing, Docker was used to perform the required tasks within isolated containers, ensuring the accuracy and security of the results. Docker is an open-source technology that allows the creation and management of isolated, portable environments known as containers. These containers include the software and its dependencies, ensuring consistent performance regardless of the underlying system [23]. Through Docker, the development of replicable environments is simplified, making it ideal for malware analysis, vulnerability testing, and other applications in the cybersecurity field [24].

Docker security is based on three main aspects: process isolation, kernel rule enforcement, and network security. It uses Linux kernel features such as namespaces and groups for isolation, while mechanisms such as SELinux, AppArmor, and Seccomp enhance the protection of the host. In the network domain, security is enhanced through TLS for secure image distribution and Docker Content Trust for verifying signed images. However, non-secure options, including a registry, can reduce protection [25]. For this reason, we verified that TLS is enabled in our project, as mentioned in the text. We ensured that Docker Daemon was not publicly accessible and implemented TLS authentication to prevent unauthorized access [26]. Finally, containers are tightly integrated into the host operating system, offering higher performance than virtual machines (VMs) and focusing on the relationship between the host and the container [25].

4.1. Unauthorized Access (Brute force attack)

A Brute Force Attack involves systematically trying all possible combinations of characters until the correct password is found. This attack is perilous when the attacker has additional information or has observed patterns

in passwords, such as common passwords or variations of the same password. It exploits weaknesses in easily guessable passwords, and defending against it requires mechanisms like multi-factor authentication and stronger access restrictions [27].

To assess the security of our system, a group of individuals tested an unauthorized access scenario by manually attempting to decode the password. These attempts were then replicated via an automated script, as shown in Figure 3 in the appendix. The script executed 51 password attempts, including 'password1', '123456', 'letmein', 'Pas5word1', and 'infelxtlttiologist5', resulting in failed login attempts. During testing, the system responded to repeated failed login attempts by restricting access from the source, demonstrating the presence of server-side protection mechanisms such as rate limiting or IP-based blocking. This was implemented through a backend script integrated with the login form, which monitors authentication failures and activates protective measures once a predefined threshold is exceeded. The attack was executed locally using Docker inside a container created specifically for this purpose.

The attack on our system was unsuccessful, as the system remained fully secure, and no data breach occurred. Had the attack been successful, it could have led to unauthorized access, exposure of sensitive data, and disruption of normal system operations. This highlights the critical need for implementing robust security mechanisms, including multi-factor authentication, strong password policies, and comprehensive access restrictions, to safeguard systems against similar threats in the future.

To explore this attack further and its link to the reckless use of antibiotics, it becomes clear that unauthorized access to falsified prescription data presents a significant risk for the health sector [28]. The misuse of antibiotics, whether through incorrect dosages or unauthorized distribution channels, accelerates microbial resistance. In regions with weak regulatory frameworks, the uncontrolled circulation of antibiotics further exacerbates the problem [29].

4.2. Denial of Service (DoS) Attacks

A Denial of Service (DoS) attack aims to exhaust resources such as CPU, memory, and network, causing disruptions to applications and system infrastructure, resulting in performance failure or application crashes [30]. More in detail, this attack is a common type of network attack, where the attacker floods the system with excessive traffic, overwhelming its resources. This results in delays or complete unavailability of the healthcare system, preventing legitimate users from accessing critical healthcare information. As part of the Security Attacks in the Network Phase, it targets the network infrastructure, impairing communication and access to essential services [27].

We executed this attack in our system, and as Figure 4 in the appendix shows, there were 62 attempts in the first test, 451 in the second, and 155 in the third, all classified as DDoS attacks. It is worth noting that as the number of attacks increases, so does the number of users trying to access the system.

The tests were conducted using the Siege tool, executed within a Docker container to ensure consistent testing conditions and isolate the tests from interference with live systems. The Docker environment provided a controlled setup, allowing us to simulate concurrent users and analyze the system's performance under various loads.

This attack concludes that our system remained operational, as indicated by the "availability," which is 100%. The results in Figure 4 in the appendix highlight the following:

- Response times of 0.15 seconds, 0.14 seconds, and 0.13 seconds;
- Transaction rates of 14.73, 11.70, and 5.21 transactions per second, respectively;
- Concurrency levels of 2.15, 1.62, and 0.70 simultaneous connections;
- Total data transferred: 0.42 MB, 3.13 MB, and 1.26 MB;
- Throughput values of 0.10 MB/s, 0.08 MB/s, and 0.04 MB/s;
- All transactions (62, 451, and 155) were successfully executed without failure, confirming the system's ability to handle increased requests efficiently.

These results validate the system's robustness, showcasing its ability to maintain 100% availability and stable performance, even during simulated DDoS attack scenarios.

4.3. Distributed Denial of Service (DDoS) Attacks

DDoS attacks often originate from networks of computers that have been infected with malicious software and are used to attack the target without being immediately detected. The severity of these attacks can be enormous, as they are often difficult to trace back to their source and can last for extended periods, disrupting the functionality of the network or system [27].

The previous section, Denial of Service (DoS) Attacks, provided a comprehensive analysis of the attack, including its behavior and impact. The DDoS attack is illustrated in Figure 4 in the appendix, which shows the second test, where 451 attempts were recorded.

4.4. SQL Injection Attempts

According to Priya et al. (2017), SQL Injection attacks involve injecting malicious SQL statements into a database query to exploit system vulnerabilities. We utilized the SQLMap tool within a Docker container to simulate and study such attacks. Running SQLMap in a controlled Docker environment ensured consistent results while preventing unintended consequences on other systems. The target web application mimicked common database configurations, allowing SQLMap to execute automated tests on various dynamic parameters. These tests helped identify potential vulnerabilities under realistic conditions.

As part of our research, we conducted an SQL Injection testing scenario. As shown in Figure 5 in the appendix, SQLMap begins the detection process by analyzing dynamic parameters, such as the User-Agent header. Although identified as dynamic, this parameter is ultimately deemed non-exploitable due to possible false positives. The tool then detects SAP MaxDB as the probable backend database management system (DBMS) and recommends focusing on specific payloads for that platform.

Further analysis includes specialized SQLMap payload testing targeting SAP MaxDB and evaluating additional parameters such as Host. However, none of these are found to be vulnerable. Despite testing multiple techniques, including Boolean-based blind, time-based blind, and UNION query injection, no exploitable entry points were detected.

SQLMap concludes with a CRITICAL message, indicating that none of the analyzed parameters are vulnerable. This suggests the potential presence of a protection mechanism, such as a Web Application Firewall (WAF), to prevent such attacks. To bypass these defenses, SQLMap suggests using the `--tamper=space2comment` or `--random-agent` options.

Successful SQL Injection attacks could lead to unauthorized access to sensitive healthcare data, including patient records, compromising privacy and data integrity. This underscores the importance of implementing robust cybersecurity measures to protect medical databases and ensure the proper use of antibiotics. Attackers exploiting such vulnerabilities could alter or delete critical medical data, posing a risk to patient safety and disrupting vital healthcare operations.

Our tests yielded positive results, as we found no exploitable parameters within the target system. The backend database, such as SAP MaxDB, offers robust security protection. However, these findings underscore the need for ongoing improvements in security measures, as sophisticated attackers may develop increasingly sophisticated techniques to circumvent existing defenses.

5. Applying Machine Learning for Securing SQL Injection

While our tests indicated strong performance, highlighting the robust security protections of the backend database, they also underscore the importance of proactive measures. Thus, to enhance our security framework, we developed an application that integrates machine learning to determine whether SQL injection attempts are safe or malicious.

We first analyzed the structure of our existing prescription submission form and the underlying SQL tables supporting its functionality. Based on this structure, we simulated realistic query inputs that a user might submit, both legitimate and malicious.

This analysis informed the creation of a balanced synthetic dataset of 500 queries, comprising 250 valid SQL statements (e.g., `SELECT * FROM prescriptions WHERE doctor_id = 5`) and 250 malicious injection attempts (e.g., `SELECT * FROM users WHERE id = 1 OR 1=1; --`).

The malicious examples were not purely idealized; we deliberately included noisy and obfuscated variations to reflect the unpredictability and complexity of real-world SQL injection attempts. This approach aimed at improving the model's generalization and robustness in practical scenarios.

Each query was transformed into a feature vector using a CountVectorizer and enhanced with custom features, including the number of special characters and the occurrences of SQL keywords. This preprocessing step was automated and saved in reusable formats (features.py, vectorizer.pkl) for consistency across all experiments.

We then performed a comparative evaluation of four machine learning models: Random Forest, Decision Tree, Support Vector Machine, and Naive Bayes. Each model was trained and tested on the same dataset and evaluated using accuracy, precision, recall, and F1-score.

As shown in Figure 6 in the appendix, the models exhibited strong performance across all metrics. Among them, the SVM and Naive Bayes models achieved an accuracy of 89%, while the Random Forest model

outperformed them with an accuracy of 94.2% and an F1-score of 0.88. This highlights its robustness even with a relatively small dataset. The Random Forest model was ultimately selected due to its strong balance between precision and recall.

This model was then retrained on the full dataset and serialized (`sql_injection_model.pkl`) along with its associated vectorizer. We developed a Flask API that loads the model and exposes a `/predict` endpoint. Incoming SQL queries are classified in real time, and a JSON response is returned with the result: "SQL Injection Detected" or "Query is Safe".

As also demonstrated in Figure 6 in the appendix, after running the Flask server (`app.py`), we submitted a known SQL injection attempt (`DROP TABLE users; --`) via the interface, and the system correctly flagged it as "SQL Injection Detected". This confirms that the model functions properly in a live environment.

To demonstrate its usability, we built a web interface using HTML and JavaScript. The interface allows system administrators to input queries manually and receive instant feedback. Additionally, we implemented an auto-checking mode, where the system sends queries for analysis at fixed time intervals (currently every 5 seconds), simulating autonomous scanning of backend query logs.

Figure 7 in the appendix shows the server-side activity during live testing. It captures successful requests to predict, which returned responses, indicating that the system was actively receiving and processing query inputs in real time.

The system supports both manual and automated testing of SQL queries through a unified web interface. The system administrator can submit individual queries for real-time analysis or activate the auto-test mode, which continuously sends queries for classification every 5 seconds, allowing both modes to operate in parallel without conflict. This flexibility enables both interactive experimentation and autonomous background monitoring.

In conclusion, the integration of machine learning into our SQL injection detection pipeline has proven both technically viable and practically effective. By embedding intelligent query analysis directly into the prescription system's backend, we enhance its security posture without compromising usability. The system performs reliably under real-time conditions and maintains consistent detection performance across a variety of input types, forming a robust core defense within the broader prescription infrastructure.

6. Results

Our system provides a safe and effective solution for monitoring antibiotic prescriptions and ensuring their proper use in prescribing and administration. It allows physicians to register and track prescriptions in a structured and secure manner.

A summary of the security tests conducted in Section 4 is provided in Table 1 below. This comparative overview presents the system's resilience under simulated cyberattack scenarios, including Brute Force, DoS, DDoS, and SQL Injection attempts. The results confirm that the system maintained its integrity and availability throughout the evaluation process. Table 1 summarizes the outcomes of the system's resilience under these simulated attacks. Each scenario was tested in a controlled Docker environment to ensure consistency and isolation. The results validate the platform's effectiveness in safeguarding sensitive healthcare data against prevalent cybersecurity threats.

Table 1: Summary of security testing results and system response under simulated cyberattacks.

Attack Type	Attempts	System Response	Testing Method	Detection Result
Brute Force	51 password attempts	All login attempts failed; attack detected and blocked	Common passwords	Attack blocked
DoS	62 concurrent requests	Average response time: ~0.15 seconds; system remained responsive	Siege tool	Attack withstood
DDoS	451 concurrent requests	100% uptime maintained; minor latency and expected throughput drop	Siege tool	Attack withstood
SQL Injection	0 exploited vulnerabilities	No injection points or exploitable parameters identified	SQLMap	Attack unsuccessful

These findings highlight the system's potential for deployment in real-world healthcare environments where secure data handling is paramount.

The form is hosted on a secure HyperText Transfer Protocol Secure (HTTPS) server, utilizing TLS 1.2 and 1.3 with modern encryption suites to ensure safe data transmission. Following the issuance and integration of the certificate, the system employs Elliptic Curve Cryptography (ECC) with a 256-bit key (Sha384withEcdsa), offering high security at a lower computational cost than traditional RSA encryption.

A comprehensive SSL security assessment confirmed adherence to best practices, including support for Forward Secrecy, mitigation of known vulnerabilities, and the implementation of strong encryption mechanisms. All cryptographic protocols comply with the latest security recommendations, ensuring effective protection against emerging threats. The system employs modern cipher suites that combine high-speed processing with strong resistance to cryptographic attacks, safeguarding sensitive medical data. Additionally, outdated protocols such as SSL 2.0, SSL 3.0, TLS 1.0, and TLS 1.1 are not supported, ensuring full compliance with modern security standards.

Strong encryption policies enhance the system's resilience and align with industry-leading security frameworks. Compliance checks verified that all connections utilize HTTPS, enhancing data integrity. Furthermore, penetration testing confirmed the system's resilience against external threats, with no detected data breaches or unauthorized modifications.

Access is restricted to a single certified physician, as the system is in a trial/research phase. This controlled environment allows us to refine security measures and optimize performance before potential expansion. All files are securely stored within the hospital's cloud infrastructure, ensuring accessibility and data protection. By facilitating the accurate tracking of antibiotic prescriptions, our system promotes responsible antibiotic use [31], helping to combat antimicrobial resistance while establishing a robust cybersecurity framework for medical data protection. Rather than simply adopting cutting-edge technologies for their own sake, our approach focuses on adapting proven solutions to the real needs and constraints of the healthcare environment, ensuring a high level of security without unnecessary complexity or cost.

To further validate the system's effectiveness, we compared its SQL injection detection component against benchmark results reported in the literature. Prior studies on machine learning-based SQL injection detection in generic web environments typically report accuracy ranging from 85% to 92% [10][11]. In contrast, our Random Forest model achieved an accuracy of 94.2% and an F1-score of 0.88, demonstrating strong performance even in a domain-specific application and under resource constraints typical of clinical systems. Unlike prior works, our model is designed for low-resource deployment within clinical prescription platforms, enabling real-time protection without disrupting system performance.

7. Conclusion and Future Work

This paper introduced an enhanced prescription system to monitor antibiotic consumption, ensuring transparency, proper data management, and security. The integration of machine learning represents the core innovation of this work, enabling proactive detection of SQL injections within a clinical prescribing environment. Implemented using HTML, CSS, and PHP, the system demonstrated robust resistance to various cyber threats, including unauthorized access, Denial of Service (DoS) attacks, Distributed Denial of Service (DDoS) attacks, and SQL injection attempts. The results show that our system is reliable for monitoring antibiotics and protecting against cyberattacks. Protecting medical data is essential for safeguarding patient privacy, supporting responsible medication use, and contributing to faster, more effective patient recovery. Currently, the prototype system supports a single authorized user for testing and evaluation purposes. Future development will include a multi-user architecture with role-based access, concurrent data handling, and secure database storage. These enhancements aim to ensure both scalability and security within real-time healthcare environments. Importantly, to date, no existing work has demonstrated the integration of a machine learning-driven SQL injection detection layer directly within the operational context of an antibiotic prescription system.

This will allow the system to scale and handle a broader range of tasks, such as monitoring the number of consultations and providing data analytics for medical professionals. Future improvements will include implementing multilingual support to accommodate diverse healthcare environments. The system could further evolve by transitioning its database into a blockchain, allowing doctors to form decentralized networks and facilitate the secure addition of new members.

As an extension of our work, integrating blockchain technology could be particularly beneficial for healthcare organizations. Furthermore, the future commercial availability of quantum computing could offer advanced encryption and processing capabilities, further enhancing the system's security and efficiency while

unlocking new possibilities. Additional security tests, including database-specific attacks and integration with Elliptic Curve Cryptography (ECC) for enhanced network protection, will be crucial for maintaining and improving system resilience. By addressing these future capabilities, the system can become even more secure, scalable, and adaptable, laying the foundation for its broader use and long-term sustainability in healthcare settings.

Author Contributions:

Conceptualization, C.K. and V.T.; methodology, S.M. and V.T.; formal analysis, S.M.; writing—original draft preparation, S.M.; writing—review and editing, S.M. and C.K.; supervision, V.T. All authors have read and agreed to the published version of the manuscript.

Funding:

This research received no external funding.

Data Availability Statement:

No real-world data were created or analyzed in this study. A synthetic dataset was generated for illustrative purposes, but it does not contain actual research data.

Conflicts of Interest:

The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AES	Advanced Encryption Standard
AHRQ	Agency for Healthcare Research and Quality
AMR	Antimicrobial Resistance
CNAME	Canonical Name
CAP	Community-Acquired Pneumonia
CSR	Certificate Signing Request
CSS	Cascading Style Sheets
DDoS	Distributed Denial of Service
DoS	Denial of Service
ECC	Elliptic Curve Cryptography
GCM	Galois/Counter Mode
HTML	HyperText Markup Language
HTTPS	HyperText Transfer Protocol Secure
PHP	Hypertext Preprocessor (originally Personal Home Page)
RSA	Rivest Shamir Adleman
SAMaxDB	SAP Maximum Database
SHA	Secure Hash Algorithm Abbreviations
SQL	Structured Query Language
SSL	Secure Sockets Layer
TLS	Transport Layer Security

References

- [1] Mariettou, S., Koutsojannis, C., & Triantafillou, V. (2024, August). Predicting Coronary Heart Disease through Machine Learning Algorithms. In *The International Conference on Innovations in Computing Research* (pp. 652-659). Cham: Springer Nature Switzerland. doi.org/10.1007/978-3-031-65522-7_56
- [2] SaberiKamarposhti, M., Ng, K. W., Chua, F. F., Abdullah, J., Yadollahi, M., Moradi, M., & Ahmadpour, S. (2024). Post-quantum healthcare: A roadmap for cybersecurity resilience in medical data. *Heliyon*, 10(10). doi.org/10.1016/j.heliyon.2024.e31406
- [3] World Health Organization. (2023, November 23). *Control antibiotic misuse or the drugs won't work, warn WHO experts*. <https://www.who.int/europe/news-room/23-11-2023-control-antibiotic-misuse-or-the-drugs-won-t-work--warn-who-experts>
- [4] European Commission. (2023). UNGA Political Declaration: A global commitment to fight antimicrobial resistance (AMR). <https://ec.europa.eu/newsroom/hera/items/852435/en>
- [5] Mariettou, S., Koutsojannis, C., & Triantafillou, V. (2024). Security Systems in Greek Health Care Institutions: A Scoping Review Towards an Effective Benchmarking Approach. In *Proceedings of the International Conferences e-Society* (pp. 53-60).
- [6] Wang, Y. W., & Wu, J. L. (2023). A Privacy-Preserving Symptoms Retrieval System with the Aid of Homomorphic Encryption and Private Set Intersection Schemes. *Algorithms*, 16(5), 244. doi: 10.3390/a16050244.
- [7] English, B. K., & Gaur, A. H. (2009). The use and abuse of antibiotics and the development of antibiotic resistance. *Hot topics in infection and immunity in children VI*, 73-82. doi: 10.1007/978-1-4419-0981-7_6.
- [8] Javaid, M., Haleem, A., Singh, R. P., & Suman, R. (2023). Towards insighting cybersecurity for healthcare domains: A comprehensive review of recent practices and trends. *Cyber Security and Applications*, 1, 100016. doi: 10.1016/j.csa.2023.100016.
- [9] Salam, M. A., Al-Amin, M. Y., Salam, M. T., Pawar, J. S., Akhter, N., Rabaan, A. A., & Alqumber, M. A. (2023, January). Antimicrobial resistance: a growing serious threat for global public health. In *Healthcare* (Vol. 11, No. 13, p. 1946). Multidisciplinary Digital Publishing Institute. doi: 10.3390/healthcare11131946.
- [10] Gupta, A., Tyagi, L. K., & Mohamed, A. (2023, November). A Machine Learning Methodology for Detecting SQL Injection Attacks. In *2023 3rd International Conference on Technological Advancements in Computational Sciences (ICTACS)* (pp. 184-191). IEEE. doi.org/10.1109/ICTACS59847.2023.10390153
- [11] Chen, D., Yan, Q., Wu, C., & Zhao, J. (2021). Sql injection attack detection and prevention techniques using deep learning. In *Journal of Physics: Conference Series* (Vol. 1757, No. 1, p. 012055). IOP Publishing. doi.org/10.1088/1742-6596/1757/1/012055
- [12] Shaikh, J. A., Wang, C., Sima, M. W. U., Arshad, M., Owais, M., Hassan, D. S. M., Alkanhel, R., & Muthanna, M. S. A. (2025). A deep Reinforcement learning-based robust Intrusion Detection System for securing IoMT Healthcare Networks. *Frontiers in Medicine*, 12, 1524286. doi.org/10.3389/fmed.2025.1524286
- [13] Roumani, Y., & Alraee, M. (2025). Examining the factors that impact the severity of cyberattacks on critical infrastructures. *Computers & Security*, 148, 104074. doi: 10.1016/j.cose.2024.104074.
- [14] Westbrook, J. I., Seaman, K., Wabe, N., Raban, M. Z., Urwin, R., Badgery-Parker, T., Mecardo, C., Mumford, V., Nguyen, A. D., Root, J., Balmer, S., Waugh, K., Pinto, S., Burge, B., Aldegue, E., Dunstan, T., Jorgensen, M., Gray, L., Bucknall, T., et al. Cumming, A. (2024). In *MEDINFO 2023—The Future Is Accessible* (pp. 404-408). IOS Press. doi.org/10.3233/shti230996
- [15] Tamma, P. D., Miller, M. A., Dullabh, P., Ahn, R., Speck, K., Gao, Y., Scherpf, E., & Cosgrove, S. E. (2021). Association of a Safety Program for Improving Antibiotic Use With Antibiotic Use and Hospital-Onset *Clostridioides difficile* Infection Rates Among US Hospitals. *JAMA Network Open*, 4(2), e210235. doi.org/10.1001/jamanetworkopen.2021.0235
- [16] Daneman, N., Rishu, A. H., Pinto, R., Aslanian, P., Bagshaw, S. M., Carignan, A., Charbonney, E., Coburn, B., Cook, D. J., Detsky, M. E., Dodek, P., Hall, R., Kumar, A., Lamontagne, F., Lauzier, F., Marshall, J. C., Martin, C. M., McIntyre, L., Muscedere, J., et al. Fowler, R. A. (2018). 7 versus 14 days of antibiotic treatment for critically ill patients with bloodstream infection: a pilot randomized clinical trial. *Trials*, 19(1). doi.org/10.1186/s13063-018-2474-1
- [17] Kuijpers, S. M. E., Buis, D. T. P., Ziesemer, K. A., Van Hest, R. M., Schade, R. P., Sigaloff, K. C. E., & Prins, J. M. (2024). The evidence base for the optimal antibiotic treatment duration of upper and lower

respiratory tract infections: An umbrella review. *The Lancet Infectious Diseases*. doi: 10.1016/s1473-3099(24)00456-0.

- [18] Mikhail, S., Singh, N. B., Kebriaei, R., Rice, S. A., Stamper, K. C., Castanheira, M., & Rybak, M. J. (2019). Evaluation of the synergy of ceftazidime-avibactam in combination with meropenem, amikacin, aztreonam, colistin, or fosfomycin against well-characterized multidrug-resistant *Klebsiella pneumoniae* and *Pseudomonas aeruginosa*. *Antimicrobial agents and chemotherapy*, 63(8), 10-1128. doi.org/10.1128/AAC.00779-19
- [19] Mariettou, S., Koutsojannis, C., & Triantafyllou, V. (2025). Artificial Intelligence and Algorithmic Approaches of Health Security Systems: A Review. *Algorithms*, 18(2), 59. doi: 10.3390/a18020059.
- [20] George, N., & Manuel, M. (2024). A secure data hiding system in biomedical images using grain 128a algorithm, logistic mapping and elliptical curve cryptography. *Multimedia Tools and Applications*, 1-24. doi: 10.1007/s11042-024-19147-2.
- [21] Farhat, S., Kumar, M., Srivastava, A., Bisht, S., Rishiwal, V., & Yadav, M. (2025). Enhancing E-Healthcare Data Privacy Through Efficient Dual Signature on Twisted Edwards Curves Encryption Decryption. *Security and Privacy*, 8(1), e464. doi: 10.1002/spy2.464.
- [22] Ivanov, O., Ruzhentsev, V., & Oliynykov, R. (2018, October). Comparison of modern network attacks on TLS protocol. In *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)* (pp. 565-570). IEEE. doi: 10.1109/INFOCOMMST.2018.8632026.
- [23] Mouat, A. (2015). *Using Docker: Developing and deploying software with containers*. " O'Reilly Media, Inc."
- [24] Alyas, T., Ali, S., Khan, H. U., Samad, A., Alissa, K., & Saleem, M. A. (2022). Container performance and vulnerability management for container security using docker engine. *Security and Communication Networks*, 2022(1), 6819002. doi/full/10.1155/2022/6819002
- [25] Combe, T., Martin, A., & Di Pietro, R. (2016). To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5), 54-62. doi: 10.1109/MCC.2016.100.
- [26] Borglund, N. (2024). Security and application deployment using Docker (Bachelor's thesis, Uppsala University). <https://www.diva-portal.org/smash/get/diva2:1913443/FULLTEXT01>
- [27] Priya, R., Sivasankaran, S., Ravisasthiri, P., & Sivachandiran, S. (2017). A survey on security attacks in electronic healthcare systems. In *2017 international conference on communication and signal processing (ICCSP)* (pp. 0691-0694). IEEE. doi: 10.1109/ICCSP.2017.8286448.
- [28] Cesaro, A., Hoffman, S. C., Das, P., & de la Fuente-Nunez, C. (2025). Challenges and applications of artificial intelligence in infectious diseases and antimicrobial resistance. *npj Antimicrobials and Resistance*, 3(1), 2. doi: 10.1038/s44259-024-00068-x.
- [29] Islam, M. F., Arka, P. B., Rohman, M., Hossain, M. S., Babu, M. R., Azhari, H. A., & Uddin, M. J. (2025). Pooling the complex survey data across the 64 lower and middle-income countries: A study on antibiotic usage in under-five children. *Heliyon*, 11(1). doi: 10.1016/j.heliyon.2024.e41470.
- [30] Haq, M. S., Nguyen, T. D., Tosun, A. Ş., Vollmer, F., Korkmaz, T., & Sadeghi, A. R. (2024, May). SoK: A comprehensive analysis and evaluation of docker container attack and defense mechanisms. In *2024 IEEE Symposium on Security and Privacy (SP)* (pp. 4573-4590). IEEE. doi: 10.1109/SP54263.2024.00268.
- [31] Compton, W. M., & Volkow, N. D. (2006). Abuse of prescription drugs and the risk of addiction. *Drug and alcohol dependence*, 83, S4-S7. doi: 10.1016/j.drugalcdep.2005.10.020.

Appendix

Figure 3. Brute-force attack Docker to detect unauthorized access.

```

Terminal
Failed login attempt with password: adminpass2025
Attempt 43: Trying URL: http://rehabsciences.42web.io/ with password: trylogin
Failed login attempt with password: trylogin
Attempt 44: Trying URL: http://rehabsciences.42web.io/ with password: lockoutme
Failed login attempt with password: lockoutme
Attempt 45: Trying URL: http://rehabsciences.42web.io/ with password: safetyfirst
Failed login attempt with password: safetyfirst
Attempt 46: Trying URL: http://rehabsciences.42web.io/ with password: try2hack
Failed login attempt with password: try2hack
Attempt 47: Trying URL: http://rehabsciences.42web.io/ with password: injectthis
Failed login attempt with password: injectthis
Attempt 48: Trying URL: http://rehabsciences.42web.io/ with password: localhost
Failed login attempt with password: localhost
Attempt 49: Trying URL: http://rehabsciences.42web.io/ with password: infex_trial
Failed login attempt with password: infex_trial
Attempt 50: Trying URL: http://rehabsciences.42web.io/ with password: P@ssw0rd
Failed login attempt with password: P@ssw0rd
Attempt 51: Trying URL: http://rehabsciences.42web.io/ with password: 0p3nAcc3ss
Failed login attempt with password: 0p3nAcc3ss
PS C:\Users\smari>

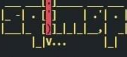
RAM 1.04 GB CPU 0.00% Disk 1016.91 GB avail. of 1081.10 GB
Terminal

```

Figure 4. Simulation of DoS and DDoS attacks on a Docker container to evaluate system resilience.

```
    "data_transferred": 0.42,
    "response_time": 0.15,
    "transaction_rate": 14.73,
    "throughput": 0.10,
    "concurrency": 2.15,
    "successful_transactions": 62,
    "failed_transactions": 0,
    "longest_transaction": 0.54,
    "shortest_transaction": 0.13
  }
root@32ccb4ecd16e:/# ^C
root@32ccb4ecd16e:/# ^C
root@32ccb4ecd16e:/# siege -c 5 -t 1M -d 1 -v http://rehabsciences.42web.io/
^C
{
  "transactions": 451,
  "availability": 100.00,
  "elapsed_time": 30.55,
  "data_transferred": 3.13,
  "response_time": 0.14,
  "transaction_rate": 11.70,
  "throughput": 0.08,
  "concurrency": 1.62,
  "successful_transactions": 451,
  "failed_transactions": 0,
  "longest_transaction": 1.14,
  "shortest_transaction": 0.12
}
root@32ccb4ecd16e:/# ^C
root@32ccb4ecd16e:/# ^C
root@32ccb4ecd16e:/# siege -c 5 -t 30s -d 1 -v http://rehabsciences.42web.io/
{
  "transactions": 155,
  "availability": 100.00,
  "elapsed_time": 29.76,
  "data_transferred": 1.07,
  "response_time": 0.13,
  "transaction_rate": 5.21,
  "throughput": 0.04,
  "concurrency": 0.70,
  "successful_transactions": 155,
  "failed_transactions": 0,
}
```

Figure 5. SQLMap detects potential injection vulnerabilities in the target system.



https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 12:42:33 /2024-12-01/

```
[12:42:33] [INFO] testing connection to the target URL
[12:42:34] [INFO] checking if the target is protected by some kind of WAF/IPS
[12:42:34] [WARNING] reflective value(s) found and filtering out
[12:42:34] [INFO] testing if the target URL content is stable
[12:42:34] [INFO] target URL content is stable
[12:42:34] [INFO] testing if parameter 'User-Agent' is dynamic
[12:42:34] [WARNING] parameter 'User-Agent' does not appear to be dynamic
[12:42:34] [WARNING] heuristic (basic) test shows that parameter 'User-Agent' might not be injectable
[12:42:34] [INFO] testing for SQL injection on parameter 'User-Agent'
[12:42:34] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[12:42:40] [INFO] parameter 'User-Agent' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable
[12:42:43] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'SAP MaxDB'
it looks like the back-end DBMS is 'SAP MaxDB'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
[12:42:43] [INFO] testing 'SAP MaxDB OR time-based blind (heavy query - comment)'
[12:42:43] [INFO] testing 'SAP MaxDB time-based blind - Parameter replace (heavy query)'
[12:42:43] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[12:42:43] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[12:42:45] [INFO] testing 'Generic UNION query (random number) - 1 to 20 columns'
[12:42:47] [INFO] testing 'Generic UNION query (NULL) - 21 to 40 columns'
[12:42:49] [INFO] testing 'Generic UNION query (random number) - 21 to 40 columns'
[12:43:31] [INFO] testing 'Generic UNION query (random number) - 81 to 100 columns'
[12:43:33] [INFO] checking if the injection point on User-Agent parameter 'User-Agent' is a false positive
[12:43:33] [WARNING] false positive or unexploitable injection point detected
[12:43:33] [WARNING] parameter 'User-Agent' does not seem to be injectable
[12:43:33] [INFO] testing if parameter 'Referer' is dynamic
[12:43:33] [WARNING] parameter 'Referer' does not appear to be dynamic
[12:43:33] [WARNING] heuristic (basic) test shows that parameter 'Referer' might not be injectable
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] Y
[12:44:54] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[12:45:04] [INFO] testing 'Generic UNION query (random number) - 1 to 10 columns'
[12:45:10] [WARNING] parameter 'Referer' does not seem to be injectable
[12:45:10] [INFO] testing if parameter 'Host' is dynamic
[12:45:10] [WARNING] parameter 'Host' does not appear to be dynamic
[12:45:10] [WARNING] heuristic (basic) test shows that parameter 'Host' might not be injectable
[12:45:10] [INFO] testing for SQL injection on parameter 'Host'
[12:47:41] [WARNING] parameter 'Host' does not seem to be injectable
[12:47:41] [RETIRED] all tested parameters do not appear to be injectable. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option
'--tamper' (e.g. '--tamper-space2comment') and/or switch '--random-agent'
```

Figure 6. Model Evaluation and Flask Testing.

```
(base) PS C:\Users\smari> cd C:\Users\smari\Desktop
(base) PS C:\Users\smari\Desktop> conda activate sql_injection_ml
(sql_injection_ml) PS C:\Users\smari\Desktop> python data.py
Dataset created with 500 balanced samples, formatted for SQL injection prescription detection.
(sql_injection_ml) PS C:\Users\smari\Desktop> python feature.py
Feature extraction completed and files saved.
(sql_injection_ml) PS C:\Users\smari\Desktop> python evaluate_models.py

Model evaluation results:
      Model  Accuracy  Precision  Recall  F1-Score
Random Forest    0.89      0.83      0.94      0.88
Decision Tree    0.89      0.82      0.94      0.87
SVM              0.89      0.83      0.94      0.88
Naive Bayes      0.89      0.83      0.94      0.88
(sql_injection_ml) PS C:\Users\smari\Desktop> python train.py
Model saved as sql_injection_model.pkl
(sql_injection_ml) PS C:\Users\smari\Desktop> python app.py
Feature extraction completed and files saved.

[INFO] Flask SQL Injection is running for testing https://rehabsciences.42web.io/
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
Feature extraction completed and files saved.

[INFO] Flask SQL Injection is running for testing https://rehabsciences.42web.io/
* Debugger is active!
```

Figure 7. Real-Time Prediction Activity via Flask API

```
127.0.0.1 - - [28/May/2025 03:04:32] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [28/May/2025 03:04:52] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:15] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:20] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:25] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:30] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:35] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:40] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:45] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:50] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:07:55] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:00] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:05] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:10] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:15] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:20] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:25] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:30] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:35] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:40] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:45] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:50] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:08:55] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2025 03:09:00] "POST /predict HTTP/1.1" 200 -
```