Counterfactual Explanation of a Classification Model for Detecting SQL Injection Attacks

Brian A. Cumi-Guzman[1,2], Alejandro D. Espinosa-Chim[1,2], Mauricio G. Orozco-del-Castillo[1,*] and Juan A. Recio-García[3]

[1]Tecnológico Nacional de México / IT de Mérida, Mérida, México
[2]AAAIMX Student Chapter at Yucatan, Mexico (AAAIMX), Association for the Advancement of Artificial Intelligence, Mexico
[3]Department of Software Engineering and Artificial Intelligence, Instituto de Tecnologías del Conocimiento, Universidad Complutense de Madrid, Spain

Abstract

In the realm of cybersecurity, accurately distinguishing between malicious and benign SQL queries is a critical challenge that impacts database security. Addressing this challenge requires advanced detection techniques capable of discerning complex patterns indicative of such attacks. This study explores the impact of syntactical modifications on SQL query classification to generate counterfactual explanation cases. By investigating how specific feature activations and deactivations influence the perceived intent of queries, the research uncovers the sensitivity of a Random Forest classifier to subtle syntactical changes. Using a counterfactuals approach, we were able to generate explanation cases that clearly identify specific features in SQL statements that are critical for detecting injection attacks. These cases evidence the importance of employing sophisticated parsing and validation techniques to accurately differentiate between potential security threats and safe database queries.

Keywords

SQL Injection Detection, Random Forest Classifier, Counterfactual Explanations, Cybersecurity

1. Introduction

In the digital era, cybersecurity stands as a primary concern, safeguarding the integrity, confidentiality, and availability of information in an increasingly interconnected world [1]. Among the variety of cyber threats, SQL injection attacks emerge as a particular challenge due to their capacity to exploit vulnerabilities in web applications to manipulate database queries [2]. These attacks not only compromise sensitive data but also undermine the foundation of trust in digital infrastructure [3]. The pervasive nature of these attacks, coupled with their potential for significant harm, remark the urgent need for robust detection mechanisms [4]. As organizations worldwide strive to fortify their defenses, developing advanced methodologies to detect and mitigate SQL injection attacks remains a critical area of research in cybersecurity [5].

Detecting SQL injection attacks presents a formidable challenge due to their diverse and sophisticated nature. These attacks exploit the dynamic execution of SQL queries, allowing attackers to manipulate database operations by injecting malicious SQL code into user inputs. The complexity of modern web applications and the subtlety of such attacks exacerbate the difficulty of detection, often requiring advanced analytical techniques beyond traditional validation and filtering methods [6]. Recent research has explored various detection strategies, including machine learning and deep neural networks, to identify patterns and anomalies indicative of SQL injection attempts [7, 8]. However, the evolving tactics employed by attackers necessitate continuous refinement of detection algorithms to maintain effectiveness [9]. The challenge is further compounded by the need to minimize false positives, which

ICCBR XCBR'24: Workshop on Case-Based Reasoning for the Explanation of Intelligent Systems at ICCBR2024, July 1, 2024, Mérida, Mexico

*Corresponding author.
le22080698@merida.tecnm.mx (B. A. Cumi-Guzman); le22080725@merida.tecnm.mx (A. D. Espinosa-Chim); mauricio.orozco@itmerida.edu.mx (M. G. Orozco-del-Castillo); jareciog@fdi.ucm.es (J. A. Recio-García)
0009-0001-9742-527X (B. A. Cumi-Guzman); 0009-0001-9754-3778 (A. D. Espinosa-Chim); 0000-0001-5793-6449 (M. G. Orozco-del-Castillo); 0000-0001-8731-6195 (J. A. Recio-García)

can disrupt legitimate database interactions, highlighting the delicate balance required in developing robust SQL injection detection mechanisms.

In our previous work [7], we investigated the explainability of several ML models for the detection of SQL injection attacks, particularly DTs, multi-layer perceptron (MLP), Random Forests (RFs), support vector machines (SVMs), ADA Boost, naive Bayes (NB), and quadratic discriminant analysis (QDA). After analyzing the most relevant features of the dataset to classify SQL attacks by these ML models, we presented a case-based explanation-by-example system. SQL sentences similar to a query were presented to the user as factual explanation cases. These explanation cases corroborated the ML model's prediction regarding the query's malicious nature.

In this paper, we move forward to investigate further explanation approaches. In the realm of machine learning, the quest for transparency and interpretability has led to the exploration of various explanatory methodologies, among which counterfactual explanations stand out for their intuitiveness and actionability [10]. Counterfactual explanations illuminate model decisions by illustrating minimal alterations to the input that would result in a different prediction, thereby offering a clear "if-then" rationale behind model outputs [11]. This approach not only demystifies the black-box nature of complex models but also aligns with legal and ethical standards, making it particularly relevant in sensitive domains like healthcare and finance [12, 13]. However, the generation of meaningful and diverse counterfactuals that account for feasibility and user context remains a challenge, necessitating further research to refine these explanations for practical applicability [14, 10]. The development of counterfactual explanations that are both understandable and actionable for end-users is crucial for bridging the gap between machine learning capabilities and human decision-making processes. In the concrete domain of case-based reasoning, there is a huge body of work regarding case-based counterfactuals and how nearest unlike neighbors can explain how a prediction might be changed [15, 16], pointing out the relevance of this explanation strategy and its relationship to CBR.

The primary objective of our research is to enhance the interpretability of machine learning models used in the detection of SQL injection attacks by leveraging counterfactual case-based explanations. Our contributions are threefold: First, we curate a comprehensive dataset of SQL injection statements [7] to encompass a broad spectrum of attack vectors, thus providing a robust foundation for model training and evaluation. Second, we employ a Random Forest classifier to distinguish between normal (non-malicious) and malicious SQL queries. Finally, we analyze the Random Forest classifier in a way that allows us to generate counterfactual explanation cases, specifically addressing the unique challenges posed by SQL injection attack detection. Our approach sheds light on the model's decision-making process while also identifying critical features that significantly influence the classification of SQL queries, thereby offering valuable insights for enhancing web application security.

"This paper is organized as follows: Section 2 explores the background of SQL injection attacks, their impact on web security, and introduces machine learning explainability and counterfactual reasoning. Section 3 describes the creation of our SQL injection dataset and the feature extraction process. Section 4 outlines the methodology of our Random forest classifier, its selection rationale, and our method for generating counterfactual explanations applicable to cybersecurity. Section 5 evaluates the model's effectiveness and the insights from counterfactual explanations. Section 6 discusses the implications of our findings and the challenges faced, considering their impact on future cybersecurity methodologies. Finally, Section 7 concludes with a summary of our results and recommendations for future research to improve SQL injection detection through machine learning explainability."

2. Background

2.1. Overview and Types of SQL Injection Attacks

SQL injection attacks represent one of the most common threats within the cybersecurity landscape, exploiting vulnerabilities in web applications to execute unauthorized SQL commands and manipulate database operations [7, 17]. These attacks leverage inadequacies in input validation, allowing attackers to inject malicious code that can lead to data breaches, unauthorized data manipulation,

and even complete database destruction [18, 19]. The gravity of SQL injection attacks is remarked by their widespread prevalence and the significant potential for harm, compromising the integrity and confidentiality of sensitive information stored in databases [20, 21]. As such, understanding the mechanics and implications of SQL injection attacks is crucial for developing effective countermeasures and safeguarding digital assets against this cybersecurity threat.

SQL injection attacks are classified into several types, each exploiting unique vulnerabilities within web applications. Classic SQL injection typically occurs when a SQL query is inserted into the application using input data from the client. Error-based SQL injection takes advantage of error messages from the database server to extract details about the database structure. On the other hand, union-based SQL injection involves utilizing the **`UNION`** SQL operator to merge the outcomes of multiple **`SELECT`** statements into one result set. Blind SQL injection, one of the more insidious forms, relies on sending a series of true or false queries to the database, determining the value of a parameter by observing the behavior of the response [22, 19]. Tautology-based attacks, where attackers insert a tautology (a statement that is always true) into a query to bypass security measures, and inference-based attacks, where attackers deduce data structure and content without direct data exfiltration, further exemplify the sophisticated methods employed by attackers [23, 24].

2.2. Impacts and Countermeasures

The impact of SQL injection attacks extends far beyond mere technical glitches, posing significant threats to organizational security, financial stability, and reputational integrity. By exploiting vulnerabilities in web applications, attackers gain unauthorized access to databases, leading to the potential exposure, alteration, or destruction of sensitive data [19, 20]. The ramifications of such breaches are multifaceted, encompassing data confidentiality breaches, integrity compromises, and undermining system availability. Notably, the repercussions of SQL injection attacks can result in dire financial losses, legal liabilities, and erosion of customer trust, particularly when personal or financial information is compromised [22, 25]. Moreover, these attacks can serve as a gateway for further exploitation, enabling attackers to escalate their privileges within the system, execute arbitrary code, or even take complete control over the affected servers, thereby magnifying the scope and scale of the damage inflicted.

In response to the escalating threats posed by SQL injection attacks, a diverse array of detection and prevention techniques has been developed, ranging from traditional input validation and parameterization to advanced anomaly detection and machine learning-based approaches. Input validation mechanisms, which scrutinize user-provided data for malicious patterns, serve as the first line of defense, albeit with limitations in detecting sophisticated injection strategies [26]. Parameterized queries, by segregating SQL query structure from user input, significantly reduce the attack surface by preventing the execution of dynamically constructed malicious SQL code [27]. Moreover, recent advancements have introduced machine learning algorithms capable of discerning normal database queries from anomalous [7], potentially malicious ones, offering a dynamic and adaptable solution to SQL injection threats [8]. Additionally, assertion-based methods, which enforce strict constraints on SQL query structure and logic, have shown promise in preemptively mitigating injection risks [28]. Despite these technological strides, the heterogeneity and evolution of SQL injection tactics require further research and development to fortify web applications against these cybersecurity threats.

2.3. Role of Machine Learning and Explainability

The advent of machine learning has significantly enhanced the detection of SQL injection attacks, providing a dynamic and adaptable approach to identifying and mitigating these threats. Machine learning models, trained on datasets comprising both non-malicious and malicious SQL queries, have demonstrated remarkable efficacy in distinguishing between legitimate user inputs and injection attempts [7, 17, 29, 30, 8]. Techniques ranging from logistic regression and decision trees to advanced neural networks have been employed, each contributing unique strengths to the detection process [29]. Notably, models utilizing artificial neural networks, such as Multilayer Perceptrons (MLP) and

Convolutional Neural Networks (CNN), have been shown to achieve high accuracy rates, effectively learning complex patterns indicative of SQL injection attacks [30, 8]. These models excel in generalizing from training data to unseen inputs, enabling them to identify novel injection strategies beyond the scope of traditional rule-based detection systems.

However, a commonly mentioned drawback in machine learning is its opaque nature, i.e., the lack of transparency when trying to explain the models' reasons for given outputs. Therefore, in the last years, the concept of explainability in machine learning has garnered significant attention, emphasizing the importance of understanding and interpreting model decisions, particularly in high-stakes domains such as healthcare, finance, and legal systems. Explainable AI (XAI) aims to bridge the gap between the predictive capabilities of complex models and the human need for comprehensible explanations of these predictions. Various methodologies have been proposed to achieve explainability, including feature importance scores, model-agnostic methods, and counterfactual explanations, each offering different insights into model behavior [31, 32]. Despite these advances, challenges remain in aligning machine-generated explanations with human intuition and ensuring these explanations genuinely reflect the underlying model rationale. Moreover, the operationalization of explainability requirements often involves a trade-off between model performance and interpretability, necessitating a careful balance to maintain the efficacy of machine learning applications [33, 34].

Counterfactual explanations have emerged as a compelling approach within the realm of XAI, offering a specific class of explanation that provides a link between what could have happened had input to a model been changed in a particular way [11]. This form of explanation is particularly appealing because it aligns with human cognitive processes, where understanding often comes from considering alternative scenarios and their outcomes; people frequently engage in imagining different outcomes from those that occurred [35] and counterfactuals help in behavior regulation and performance improvement [36]. Counterfactual explanations not only aid in making machine learning models more transparent but also empower users by providing actionable insights on how to achieve desired outcomes. For instance, in a loan approval model, a counterfactual explanation might suggest minimal changes to the applicant's profile that could lead to approval, thereby demystifying the model's decision process and highlighting pathways for recourse [37]. Such explanations have shown promise across various domains, including finance and healthcare, where they enhance trust and enable users to interact more effectively with machine learning systems [11, 14].

3. Dataset and Model

Our study began with a comprehensive dataset containing 30,876 SQL statements of both natural language and SQL queries [38]. Our initial task was to refine this dataset to focus solely on SQL injection detection, which led us to remove unrelated SQL statements. This refinement process resulted in a more focused dataset of 22,931 samples, categorized into malicious and non-malicious SQL sentences.

To further prepare our dataset for analysis, we calculated feature vectors for each SQL sentence based on 82 predefined attributes relevant to SQL injection patterns [7, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48] (Table 1). However, this transformation revealed a significant number of duplicate feature vectors within our dataset. Recognizing the importance of maintaining a diverse set of sentences for subsequent counterfactual explanations, we opted against removing all duplicates indiscriminately. Instead, we implemented a strategy to retain a controlled number of duplicates for each unique feature vector, ensuring a balance between diversity and redundancy. Specifically, we limited the occurrence of any duplicate feature vector to no more than 100 SQL statements in the non-malicious class and no more than 38 in the malicious class. This approach was carefully designed to balance the dataset, resulting in 3,246 feature vectors for each class. The elimination of sentences which yielded duplicate feature vectors was performed randomly. Additionally, feature normalization was applied to the dataset to standardize the range of the feature values. This step was particularly important given the diverse nature of the attributes, which ranged from binary indicators of specific keywords to counts of certain syntactical elements. Normalization ensured that no single feature dominated the model's learning due

Table 1
The 82 features collected from the literature which were used to classify SQL queries [7]

Feature	Description / Keyword
F1	select
F2	union
F3	update
F4	set
F5	alter
F6	where
F7	like
F8	from
F9	table
F10	database
F11	drop
F12	delete
F13	insert
F14	And | Or
F15	null
F16	=
F17	information_schema
F18	user
F19	version
F20	load_file
F21	save
F22	! | # | % | $ | NUL | SOH | STX | ETX | EOT | @
F23	&
F24	|
F25	,
F26	;
F27	\
F28	+ | - | * | /
F29	commit | rollback | grant | revoke | declare | remove
F30	count(;)
F31	/* | */
F32	\x
F33	\u
F34	connection
F35	xor
F36	inner join
F37	file | load_file | load_data_infile | into_outfile | into_dumpfile
F38	OS | Operative System | exec
F39	count(STRING)
F40	count(SUBSTR)
F41	count(SUBSTRING)
F42	count(MID)
F43	count(ASC)
F44	count(<)
F45	count(>)
F46	count(")
F47	count("")
F48	exists
F49	floor
F50	rand
F51	group
F52	order
F53	lenght
F54	ascii
F55	if
F56	count
F57	sleep
F58	between
F59	values
F60	delay
F61	wait
F62	benchmark
F63	indentity_insert
F64	truncate table
F65	username | password
F66	user | pass
F67	")"
F68	limit
F69	concat
F70	ne
F71	find
F72	eq | gt | gte | lt | it | lte | ite | in | nin
F73	mod | regex | text
F74	return
F75	return true | return 1
F76	exists
F77	create
F78	show
F79	collection
F80	while(loop)
F81	hidden equality expression
F82	large expression

to its scale.

The choice of the Random Forest model for detecting SQL injection attacks was driven by several considerations. First, its proficiency in handling high-dimensional, given our dataset's complexity, featuring 82 attributes per SQL statement. Random Forests, known for their robustness, effectively manage the intricacies of our feature-rich dataset without being particularly sensible to overfitting [49], thanks to their ensemble approach that aggregates predictions from numerous decision trees [50]. Moreover, the model's ability to discern both linear and nonlinear patterns ensures comprehensive analysis, capturing the diverse manifestations of SQL injection attacks [7]. This versatility, coupled with the model's relative interpretability—allowing insight into feature importance—makes Random Forest a valuable tool in cybersecurity contexts. It not only aids in accurate classification but also enhances understanding of underlying attack vectors.

The Random Forest classifier was trained on 80% of our dataset; the remaining 20% of the data served as a test set to evaluate the model's performance. Configured with 100 decision trees, our Random Forest model underwent an evaluation process that yielded highly accurate results. The model achieved an accuracy of 97.3%, with a precision of 97.9% and a recall of 96.6%, culminating in an F1 score of 97.2%. These metrics indicate not only the model's high success rate in correctly classifying SQL statements but also its balanced performance in terms of both positive predictive value and sensitivity. Further

Table 2
Performance Evaluation Metrics for the Random Forest Model in SQL Injection Detection

Metric	Score
Overall Accuracy	0.973
Precision (Positive Predictive Value)	0.979
Recall (True Positive Rate)	0.966
F1 Score (Harmonic Mean of Precision and Recall)	0.972
Area Under the Receiver Operating Characteristic Curve	0.993
Area Under the Precision-Recall Curve	0.989
Logarithmic Loss	0.161

bolstering our confidence in the model's discriminative power, the Receiver Operating Characteristic - Area Under the Curve (ROC-AUC) score reached 99.3%, and the Precision-Recall AUC stood at 98.9%, both metrics showcasing the model's ability to distinguish between classes under varying thresholds. The log loss, a measure of uncertainty in the predictions, was notably low at 0.16, suggesting high confidence in the predicted probabilities. In terms of specific outcomes, the model accurately predicted 648 non-malicious items, with a misclassification of 13 items as malicious. Similarly, it successfully identified 616 malicious items, with only 22 instances mislabeled as non-malicious.

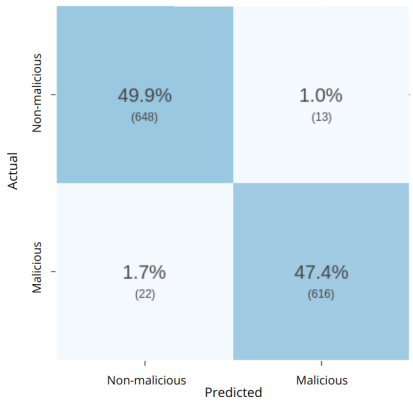


Figure 1: Confusion Matrix of the Random Forest Model. This matrix illustrates the accuracy of the model in identifying SQL queries as either malicious or non-malicious. The top-left and bottom-right cells show the number of correct predictions for non-malicious and malicious queries, respectively, while the top-right and bottom-left cells indicate the instances of incorrect classifications.

4. Methodology for the generation of Counterfactual Explanation Cases

We assume a twin-system context [51], where the opaque Random Forest model classifying SQL statements as either malicious or non-malicious is explained by finding counterfactual case-based explanations from a twinned CBR. To present these counterfactuals explanations, we are first able to present factual examples supporting the model's decisions.

There are different strategies for the generation of the counterfactual explanation cases, for example in [16] a new case-based approach for generating counterfactuals is proposed. This approach used novel ideas about the counterfactual potential and explanatory coverage of a case-base. In our case, we will follow a systematic approach following the insights of well-established algorithms such as SHAP or Feature Importance [52]. This involved systematically changing each feature's state within the input vectors, from '1' indicating presence to '0' indicating absence, and vice versa. Starting from

the model's initial predictions, we meticulously altered each feature's state in the vector and observed the model's response to each change. By adopting this iterative strategy, we were able to quantify the extent to which single-feature modifications could sway the model's judgment. Remarkably, our findings revealed that altering just one characteristic could lead a substantial percentage of cases to shift classifications—illuminating the critical role certain features play in the delineation between malicious and non-malicious SQL statements.

We refined the process of iterative feature modification to not only gauge the individual influence of each feature on the Random Forest model's predictions but also to unveil the synergistic effects that pairs of features may have on classifications. This involved an exploration of how the interplay between feature pairs within the input vectors affects classification outcomes. For each instance, we methodically altered one feature and then proceeded to iterate through the remaining features, adjusting them one by one. This comprehensive combinatorial approach enabled us to evaluate the collective impact of dual feature modifications on the model's predictions across all 82 attributes, offering a perspective on the interactions that shape classification decisions. Altering combinations of just two features resulted in a classification change for every case examined, underscoring the complex and interconnected nature of feature influences within the model and the associated case base.

In light of the influence observed when modifying combinations of feature pairs, we decided to narrow our focus to the impact of single feature alterations. This decision allowed us to concentrate on identifying the most influential individual features that sway the model's decision between classifying SQL statements as malicious or non-malicious. This focused analysis revealed significant insights into the model's sensitivity to specific features, highlighting those with the most substantial impact on classification outcomes. These results are presented in the following section.

The culmination of our counterfactual explanation methodology is illustrated through targeted case studies and the utilization of the Explainer Dashboard, both of which highlight the practical applications and real-world relevance of our findings. By presenting specific explanation cases where feature alterations led to changes in the model's classifications, we provide concrete examples that not only validate our analytical approach but also demonstrate its applicability in identifying and mitigating SQL injection threats.

These case studies delve into cases from our case base, showcasing how the activation or deactivation of key features influenced the classification outcomes. For example, the presence of certain SQL keywords or structures, previously identified as influential through our top feature analysis, can significantly alter a sentence's perceived intent, shifting it from non-malicious to malicious or vice versa. These illustrative cases serve as a bridge between the theoretical underpinnings of our methodology and its practical implications.

Complementing these case studies, the Explainer Dashboard plays a crucial role in enhancing the interpretability and transparency of our Random Forest model. By visualizing the relationship between features and classification outcomes, the dashboard facilitates a deeper understanding of the model's decision-making process. Tools such as the ROC-AUC curve and precision-recall curve within the dashboard not only validate the model's performance but also provide an intuitive platform for exploring the impact of feature modifications on classification accuracy.

5. Evaluation of the counterfactual explanation cases

In our evaluation, the initial phase of the feature impact analysis provided insights into how modifications of individual characteristics significantly affect the model's classification outcomes. Remarkably, altering just one feature shifted 87.62% of initially non-malicious cases to a malicious classification, illustrating the profound sensitivity of the model to specific feature changes. Conversely, a similar modification led to 42.85% of malicious cases being reclassified as non-malicious. As mentioned in the previous section, further investigation into the effects of simultaneously modifying two features revealed that such changes invariably resulted in classification alterations for all cases examined. This finding led us to steer our focus toward the impacts of single-feature modifications.

In our analysis, we delineated four distinct groups based on the activation and deactivation of certain features, each influencing the model's ability to classify SQL statements as malicious or non-malicious. The first group emerged from the activation of features that led to the reclassification of non-malicious cases as malicious, highlighting the sensitivity of the model to certain patterns within benign queries. The second group was defined by the activation of features that transformed malicious cases into non-malicious ones, indicating a strong association with legitimate SQL activities. In contrast, the third group involved the deactivation of features, resulting in the misclassification of non-malicious cases as malicious, emphasizing the importance of these features in affirming the non-threatening nature of SQL statements. The fourth and final group was characterized by the removal of features that led to the reclassification of malicious cases as non-malicious, highlighting their role in identifying potentially harmful queries. These classes describe the relationship between feature states and classification outcomes, providing valuable insights into the model's dynamics in detecting SQL injection attacks.

Within the first group, where feature activation leads non-malicious cases to be classified as malicious, Feature 22 (F22) plays a pivotal role, affecting 16.50% of such cases. This feature corresponds to the presence of special characters within SQL queries (1), a common hallmark of SQL injection attacks. Its activation shows the model's sensitivity to these characters, often exploited in attack vectors to manipulate or bypass database security mechanisms. The significant percentage of cases impacted by the activation of F22 highlights the critical nature of these special characters in the model's assessment of potential threats. This finding further confirms the importance of vigilance in monitoring and sanitizing user inputs that contain such characters, as they are indicative of potentially malicious intentions within the realm of SQL queries, thereby helping to bolster web application defenses against SQL injection attacks.

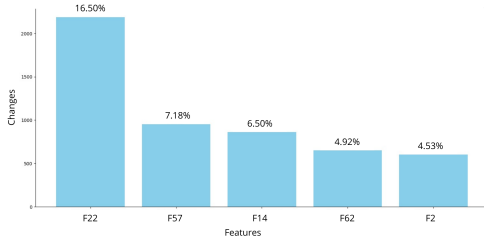
In the second group, where the activation of features transitions malicious cases to non-malicious, F52 stands out prominently. It influences 11.25% of cases, with its activation denoting the presence of the keyword **ORDER** in SQL statements. This keyword's inclusion is often indicative of legitimate database queries, particularly those involving sorting operations. The substantial impact of F52's activation on reclassifying malicious cases as non-malicious highlights its association with benign SQL queries and its role as a strong indicator of non-malicious intent. This insight is particularly valuable in refining SQL injection detection models to reduce false positives, ensuring that legitimate database interactions involving sorting are not mistakenly flagged as potential threats, thereby enhancing the accuracy and reliability of cybersecurity measures against SQL injection attacks.

For the third class, characterized by the deactivation of features leading non-malicious cases to be mistakenly identified as malicious, F8 is crucial. This feature, associated with the absence of the keyword **FROM**, affects 7.88% of such cases. The **FROM** keyword is fundamental to SQL syntax, typically used to specify the database table from which to retrieve data. Its absence, indicated by the deactivation of F8, can significantly alter the perceived intent of an SQL query, leading the model to classify queries as potential threats. This highlights the essential role of basic SQL syntax in distinguishing between malicious and non-malicious queries and the importance of considering syntactical elements in developing and refining models for SQL injection detection.

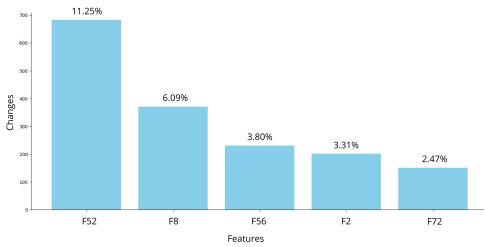
In the fourth group, where the deactivation of features shifts malicious cases to being classified as non-malicious, F16 emerges as critical, impacting 9.54% of such cases. This feature is tied to the absence of the '=' symbol, a fundamental operator in SQL often used in conditions to compare values. The removal of this symbol, suggested by the deactivation of F16, can significantly diminish the perceived maliciousness of an SQL query, as it might remove key conditions used in SQL injection attacks to exploit vulnerabilities. The substantial influence of F16's deactivation on the model's reclassification of cases shows the '=' symbol's significance in the identification of SQL injection patterns. This finding emphasizes the need for machine learning models to understand the context and usage of common SQL operators, ensuring that their absence, particularly in potentially malicious queries, does not lead to a decrease in detection accuracy, thereby enhancing the model's capability to effectively identify and mitigate SQL injection threats.

Through this analysis, it became evident that even minor modifications to specific features could lead to substantial shifts in the model's classifications, highlighting the intricate balance between malicious

and non-malicious determinations. Notably, while various features contribute to transitions across the four different groups, the impact of the primary feature in each category is considerably more pronounced than that of others. This significant disparity in influence is clearly illustrated in Figure 2, which presents the bar graphs of feature activations, and Figure 3, showcasing the effects of feature deactivations. Furthermore, Table 3 compiles a quantitative summary, incorporating the five most influential features from each of the four identified groups to illustrate the percentage impact of both activation and deactivation on classification transitions.

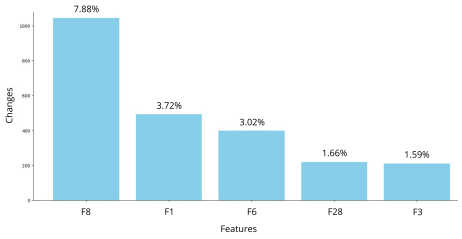


(a) Features activation leading to malicious classification

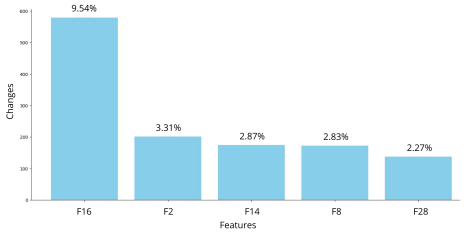


(b) Features activation leading to non-malicious classification

Figure 2: Impact of Feature Activation on SQL Statement Classification: (a) illustrates the top 5 features that, when activated, most frequently lead to an SQL statement being classified as malicious. (b) depicts the top 5 features that, upon activation, are most likely to result in a classification shift to non-malicious.



(a) Features deactivation leading to malicious classification



(b) Features deactivation leading to non-malicious classification

Figure 3: Impact of Feature Deactivation on SQL Statement Classification: (a) demonstrates the top 5 features that, when deactivated, most commonly result in an SQL statement being classified as malicious. (b) shows the top 5 features that, upon deactivation, most frequently lead to a reclassification as non-malicious.

Building on this analysis, we proceeded to a counterfactual exploration designed to delve into the dynamics of feature activation and deactivation within real SQL queries belonging to our dataset. We present four different cases, each one corresponding to one of the different groups identified before 1) activation of F22 switching from a non-malicious query to a malicious one; 2) activation of F52 switching from malicious to non-malicious; 3) deactivation of F8 switching from non-malicious to malicious; and 4) deactivation of F16 switching from malicious to non-malicious.

In the evaluation of the first group, the one where the activation of a feature leads to a malicious classification, our focus centers on the activation of feature F22, which pertains to the inclusion of specific symbols within SQL queries, such as the '#' character. For instance, consider a benign query intended for routine user authentication: `SELECT * FROM users WHERE username = 'admin' AND password = 'password123';`. Upon the activation of F22, the insertion of a '#' symbol transforms this query into `SELECT * FROM users WHERE username = 'admin'# AND password = 'password123';`, effectively neutralizing the password check by treating the remainder of the query as a comment. This alteration not only undermines the security mechanism intended to authenticate user access but also illustrates the ease with which a benign query can be manipulated into a tool

Table 3
Analysis of the Five Most Influential Features from Each of the Four Classes Due to Activation and Deactivation on Classification Transitions from both Malicious to Non-malicious (M2NM) and Non-malicious to Malicious (NM2M)

Feature	Feature Description	Activation Impact (%)	Deactivation Impact (%)
F22	Special Characters	NM2M (16.50%)	-
F56	**COUNT**	NM2M (7.18%)	M2NM (3.80%)
F14	And | Or	NM2M (6.50%)	M2NM (2.87%)
F62	BENCHMARK	NM2M (4.92%)	-
F2	**UNION**	NM2M (4.53%)	M2NM (3.31%)
F52	**ORDER**	-	M2NM (11.25%)
F8	**FROM**	-	NM2M (7.88%), M2NM (2.83%)
F72	Comparison Operators	-	M2NM (2.47%)
F1	**SELECT**	-	NM2M (3.72%)
F6	**WHERE**	-	NM2M (3.02%)
F28	Arithmetic Operators	-	NM2M (1.66%), M2NM (2.27%)
F3	**UPDATE**	-	NM2M (1.59%)
F16	"="	-	M2NM (9.54%)

for unauthorized data access. This emphasizes the critical need for robust parsing and validation mechanisms in SQL query handling to preempt such subtle yet significant security vulnerabilities.

In the examination of the second group, where the activation of a feature transitions a query from malicious to non-malicious classification, our attention is drawn to Feature 52 (F52), associated with the inclusion of the **ORDER** clause. Consider a query that might raise suspicions due to its straightforward extraction of data: **SELECT** `id, name, bank_balance` **FROM** `bank_accounts` **WHERE** `account_type = 'private';`. Without specifying an order, this query could be perceived as an attempt by an attacker to rapidly harvest data, prioritizing speed over the structure of the retrieved information, which could be indicative of malicious intent. However, with the activation of F52, the introduction of **ORDER BY** `bank_balance` transforms the query into **SELECT** `id, name, bank_balance` **FROM** `bank_accounts` **WHERE** `account_type = 'private'` **ORDER BY** `bank_balance;`, suggesting a more deliberate and legitimate use of the data, such as for financial analysis or account management by authorized personnel. Including an ordering directive implies a specific analytical or operational need, reflecting the detailed, purposeful query structure typical of legitimate database interactions. This change not only mitigates concerns about the query's intent but also shows the importance of contextual cues in SQL syntax for distinguishing between benign and potentially harmful activities. It highlights the essential role of parsing and validation in SQL query evaluation to accurately discern user intent and prevent the misclassification of legitimate database operations as security threats.

In our exploration of the third group, where the deactivation of a feature switches a query into a malicious classification, we delve into the effects of deactivating feature F8. This feature corresponds to the presence of the **FROM** keyword within SQL queries. The deactivation of F8 is illustrated through a transition from a benign query, **SELECT** `breathing (s)` **FROM** `blood` **UNION**, to one that omits the **FROM** clause, becoming **SELECT** `breathing (s)` **AS** `breathing` **UNION**. This modification, characterized by the absence of **FROM**, signals a deviation towards a syntactically irregular and potentially malicious structure. The incomplete statement, terminating abruptly with the **UNION** keyword, is particularly indicative of an attempt to exploit the query for malicious ends, such as SQL injection. This exposes a critical vulnerability, where the lack of syntactical completeness not only disrupts the logical integrity of the query but also opens a vector for security breaches.

With respect to the fourth group, where the deactivation of a feature results in a non-malicious classification, we focus on the implications of deactivating feature F16. This feature is associated with the equality operator '=' within SQL queries. The impact of F16's deactivation is illustrated by altering a query that initially might raise suspicions of malicious intent (and which is classified as so by the Random Forest clas-

sifier) due to its peculiar structure: (**SELECT CASE WHEN** (3348 = 1710) **THEN** 3348 **ELSE CAST** (1 **AS INT**) / (**SELECT** 0 **FROM** dual) **END FROM** dual). By deactivating F16 and thereby replacing the '=' operator with a '<' operator, the query is transformed into (**SELECT CASE WHEN** (3348 < 1710) **THEN** 3348 **ELSE CAST** (1 **AS INT**) / (**SELECT** 0 **FROM** dual) **END FROM** dual), mitigating the risk of division by zero and removing the potentially harmful operation as long as the "dual" table is present in the database and accessible. This alteration highlights specific operators' significance and roles in SQL query classification. The removal of the equality check in favor of a comparative operation changes the query's context and interpretation, distancing it from patterns commonly associated with SQL injections or other forms of database attacks.

The four different cases are described in Table 4.

Table 4
Counterfactual Analysis of SQL Query Modifications: This table presents a comparative analysis of SQL query modifications, showcasing how specific feature activations (F22, F52) or deactivations (F8, F16) can alter a query's classification from malicious to benign and vice versa. Each entry illustrates the impact of syntactical changes on the perceived intent of the query, highlighting the sensitivity of SQL query classifiers to such modifications.

Feature	Malicious	Non-malicious	Explanation
F22 (Special Characters including #)	`SELECT * FROM users WHERE username = 'admin' # AND password = 'password123';`	`SELECT * FROM users WHERE username = 'admin' AND password = 'password123';`	The use of # in the malicious example effectively comments out the password condition, enabling unauthorized access by bypassing authentication checks. In contrast, the benign example represents a standard authentication query without alterations.
F52 (**ORDER**)	`SELECT id, name, bank_balance FROM bank_accounts WHERE account_type = 'private';`	`SELECT id, name, bank_balance FROM bank_accounts WHERE account_type = 'private' ORDER BY bank_balance;`	The absence of the ORDER BY clause in the malicious query suggests an indiscriminate data retrieval attempt, typical of unauthorized extraction. The inclusion of ORDER BY in the benign query signifies a structured and legitimate data review or analysis.
F8 (**FROM**)	`SELECT breathing ( s ) AS breathing UNION`	`SELECT breathing ( s ) FROM blood UNION`	In the malicious example, the absence of the FROM keyword indicate a syntactical manipulation aimed at SQL injection. Conversely, the legitimate use of FROM in the benign example signifies a proper UNION operation within a valid SQL query structure.
F16 (=)	`(select (case when (3348 = 1710) then 3348 else cast (1 as int) / (select 0 from dual) end) from dual)`	`(select (case when (3348 < 1710) then 3348 else cast (1 as int) / (select 0 from dual) end) from dual)`	The malicious query employs a false comparison and division by zero, indicative of an exploit attempt through error induction. The non-malicious counterpart uses a comparison that avoids such risky operations, maintaining the query's integrity.

6. Discussion

In terms of the machine learning classifier, the role of the Random Forest classifier emerges as pivotal in our analytical framework. Configured with 100 decision trees and rigorously trained on a substantial subset of our dataset, this model served to show the dynamics of SQL syntax and semantics with remarkable clarity. The classifier's performance metrics——accuracy of 97.3%, precision of 97.9%, recall of 96.6%, and an F1 score of 97.2%——clearly reflect its efficacy in distinguishing between malicious and

non-malicious queries. The high ROC-AUC score of 99.3% and Precision-Recall AUC of 98.9%, along with a notably low log loss of 0.16, reflect not only the model's accuracy but also its balanced approach in sensitivity and specificity, a critical aspect in the domain of cybersecurity where the cost of false positives and negatives carries significant weight.

Before discussing the implications of feature modifications within specific groups, it is important to appreciate the broader impact these alterations can have on the classification of SQL queries. The sensitivity of the Random Forest classifier is critical, as even the most minute changes in a query's structure can precipitate a significant shift in its classification from non-malicious to malicious or vice versa. This sensitivity to syntactical differences is indicative of the classifier's potential utility in real-world applications, where the ability to accurately identify malicious intent amidst a vast array of legitimate queries is fundamental. This supports the importance of developing and refining machine learning models that are not only adept at handling the complexity and variability of SQL syntax but also capable of adapting to evolving patterns of cyber threats. This is further confirmed by the compounded impact of modifying two features simultaneously, resulting in classification alterations for all examined cases.

The insights derived from evaluating feature modifications across different groups shed light on potential pathways for refining SQL injection detection models and improving SQL code structuring practices. The activation of feature F22, where the inclusion of a single symbol like '#' can transform a benign query into one classified as malicious, highlights the critical importance of developing sophisticated parsing and validation mechanisms. Such mechanisms must be adept at distinguishing between legitimate syntactical constructs and those potentially manipulated for unauthorized access, thereby enhancing the precision of SQL injection detection models. Conversely, the activation of feature F52, which leads to the classification of a query as non-malicious due to the inclusion of an **ORDER** clause, shows the importance of contextual understanding in SQL syntax. This scenario illustrates how certain keywords, often used in legitimate database operations, can serve as strong indicators of benign intent. Detection models can be refined to recognize and weigh such contextual cues more heavily, reducing the likelihood of false positives and ensuring that overly stringent security measures do not hinder legitimate operations.

Furthermore, the deactivation scenarios within groups three and four reveal the delicate balance between syntactical completeness and the potential for misclassification. The absence of fundamental SQL elements like the **FROM** keyword or the equality operator '=' can lead to significant shifts in query classification, emphasizing the need for SQL injection detection models to consider both the presence and absence of key syntactical features. This understanding can inform the development of carefully crafted detection algorithms that account for the complex interplay of various SQL components.

These findings also advocate for better-structured SQL code, where adherence to best practices in syntax and the mindful inclusion of key features can both facilitate legitimate database operations and mitigate the risk of unintentional security vulnerabilities. By fostering a deeper understanding of how specific syntactical choices impact query classification, database administrators and developers can write SQL code that is not only efficient but also inherently more secure against potential injection attacks. Continuous research and adaptation in these areas are essential to keep pace with evolving threats and sophisticated attack techniques, ensuring the integrity and security of database systems in an increasingly digital world.

This counterfactual scenario highlights the subtleties involved in the automated detection of SQL injections, where the absence or manipulation of fundamental SQL operators can drastically alter the classification outcome. It highlights the necessity for detection models to not only recognize the presence of potentially harmful patterns but also understand the implications of their absence. This insight is crucial for refining security mechanisms, ensuring they can effectively discern between actual threats and benign queries that may inadvertently resemble malicious patterns due to syntactical omissions or errors.

While our counterfactual analysis has provided valuable insights into the classification of SQL queries, it is imperative to acknowledge certain limitations that accompany our study. The scope of features analyzed, though comprehensive, may not encapsulate the entire spectrum of syntactical elements

relevant to SQL query intent, potentially overlooking some that could further refine classification accuracy. Additionally, the dataset employed, although robust, represents a snapshot that may not fully capture the evolving nature of SQL injection tactics or the diversity of legitimate query constructions, possibly affecting the generalizability of our findings. While effective, the use of the Random Forest model invites consideration of other machine learning models that could offer different perspectives on feature importance and classification dynamics. Future research could benefit from exploring a variety of models, such as deep learning approaches, which might uncover additional layers of complexity within SQL query analysis. Expanding the feature set to include more granular syntactical and semantic elements could further enhance the model's ability to discern between malicious and non-malicious intents. Investigating the interplay between features in greater depth may reveal more intricate patterns that contribute to the classification process. Applying the insights gained from this study to develop more sophisticated SQL injection detection and prevention tools represents a promising avenue for future work.

7. Conclusions

Our comprehensive investigation into SQL query classification consisted of both a counterfactual analysis and an evaluation of a Random Forest classifier to delve into the dynamics of detecting SQL injection attacks. Central to our discoveries is the model's sensitivity to syntactical modifications, such as the inclusion of special characters or clauses such as **ORDER** or **FROM**, which can dramatically alter query classification. These seemingly minor alterations, backed up by the model's robust performance metrics, highlight the critical role of specific SQL keywords and structures in the delineation between malicious and benign intents. The fusion of these insights with feature impact analysis and counterfactual explanations enriches our understanding of the subtleties involved in SQL injection detection. This multifaceted approach not only reveals the delicate balance required in maintaining database functionality while safeguarding against security threats but also demonstrates the profound potential of machine learning models to enhance cybersecurity defenses.

The practical implications of our findings are directly related to database security, advocating for the deployment of advanced parsing and validation mechanisms attuned to the intricacies of SQL syntax. This enhanced understanding encourages the development of security systems proficient in warding off malicious intrusions and accurately identifying legitimate database queries, thereby minimizing false positives that could interrupt essential operations. Such an approach is vital for upholding the integrity and functionality of database systems, ensuring that protective measures support rather than hinder operational efficiency. Moreover, the identification of key features influencing the model's decision-making equips cybersecurity professionals with a refined toolkit for monitoring potential threats and adjusting detection algorithms accordingly. This targeted strategy, enriched by the actionable insights from counterfactual explanations, enables a more precise calibration of web application defenses, improving the detection of SQL injection vulnerabilities while mitigating the risk of erroneous classifications. Collectively, these insights not only highlight the importance of contextual and structural awareness in SQL query evaluation but also illustrate the potential of our findings to fortify cybersecurity defenses against the sophisticated and evolving threats posed by SQL injection attacks.

Exploring a broader array of machine learning models could provide deeper insights into the classification of SQL queries, while expanding the feature set considered in analyses might uncover new dimensions of query intent. Moreover, applying the findings of this study to the development and refinement of SQL injection detection tools holds promise for significantly bolstering database defenses. Such advancements are essential not only for enhancing the precision of security measures but also for ensuring their adaptability to the sophisticated and ever-changing tactics employed by cyber adversaries. Collaboration between academia and industry practitioners will be pivotal in translating these insights into practical, robust solutions that safeguard critical data assets against emerging threats.

Acknowledgments

This work is supported by the PERXAI project PID2020-114596RB-C21 funded by MCIN/AEI/10.13039/501100011033.

References

[1] F. Özsungur, Business Management and Strategy in Cybersecurity for Digital Transformation, Handbook of Research on Advancing Cybersecurity for Digital Transformation (2021). doi:10.4018/978-1-7998-6975-7.ch008.

[2] B. Gogoi, T. Ahmed, A. Dutta, Defending against SQL Injection Attacks in Web Applications using Machine Learning and Natural Language Processing, in: 2021 IEEE 18th India Council International Conference (INDICON), IEEE, 2021, pp. 1–6. doi:10.1109/INDICON52576.2021.9691740.

[3] W. Trappe, J. Straub, Cybersecurity: A New Open Access Journal, Cybersecurity (2018). doi:10.3390/CYBERSECURITY1010001.

[4] T. Untawale, Importance of Cyber Security in Digital Era, International Journal for Research in Applied Science and Engineering Technology (2021). doi:10.22214/ijraset.2021.37519.

[5] N. Sklavos, In the Era of Cybersecurity: Cryptographic Hardware and Embedded Systems (2019). doi:10.1109/MECO.2019.8760015.

[6] L. Yu, S. Luo, L. Pan, Detecting SQL Injection Attacks based on Text Analysis, in: Proceedings of the 3rd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2019), 2019. doi:10.2991/ICCIA-19.2019.14.

[7] J. A. Recio-García, M. G. Orozco-del-Castillo, J. A. Soladrero, Case-based explanation of classification models for the detection of SQL injection attacks, in: L. Malburg, D. Verma (Eds.), Proceedings of the Workshops at the 31st International Conference on Case-Based Reasoning (ICCBR-WS 2023) co-located with the 31st International Conference on Case-Based Reasoning (ICCBR 2023), Aberdeen, Scotland, UK, July 17, 2023, volume 3438 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 200–215. URL: https://ceur-ws.org/Vol-3438/paper_15.pdf.

[8] P. Tang, W. Qiu, Z. Huang, H. Lian, G. Liu, Detection of SQL injection based on artificial neural network, Knowl. Based Syst. 190 (2020) 105528. doi:10.1016/j.knosys.2020.105528.

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

[10] F. Cheng, Y. Ming, H. Qu, DECE: Decision Explorer with Counterfactual Explanations for Machine Learning Models, IEEE Transactions on Visualization and Computer Graphics 27 (2020) 1438–1447. doi:10.1109/TVCG.2020.3030342.

[11] S. Verma, J. P. Dickerson, K. E. Hines, Counterfactual Explanations for Machine Learning: A Review, ArXiv abs/2010.10596 (2020).

[12] R. K. Mothilal, A. Sharma, C. Tan, Explaining machine learning classifiers through diverse counterfactual explanations, in: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, 2020. doi:10.1145/3351095.3372850.

[13] R. R. Fernández, I. M. de Diego, V. Aceña, A. Fernández-Isabel, J. M. Moguerza, Random forest explainability using counterfactual sets, Inf. Fusion 63 (2020) 196–207. doi:10.1016/j.inffus.2020.07.001.

[14] R. K. Mothilal, A. Sharma, C. Tan, Explaining machine learning classifiers through diverse counterfactual explanations, in: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, 2020. doi:10.1145/3351095.3372850.

[15] B. Smyth, M. T. Keane, A few good counterfactuals: Generating interpretable, plausible and diverse counterfactual explanations, in: M. T. Keane, N. Wiratunga (Eds.), Case-Based Reasoning Research and Development - 30th International Conference, ICCBR 2022, Nancy, France, September 12-15, 2022, Proceedings, volume 13405 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 18–32. URL: https://doi.org/10.1007/978-3-031-14923-8_2. doi:10.1007/978-3-031-14923-8_2.

[16] M. T. Keane, B. Smyth, Good counterfactuals and where to find them: A case-based technique for generating counterfactuals for explainable AI (XAI), in: I. Watson, R. O. Weber (Eds.), Case-Based Reasoning Research and Development - 28th International Conference, ICCBR 2020, Salamanca, Spain, June 8-12, 2020, Proceedings, volume 12311 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 163–178. URL: https://doi.org/10.1007/978-3-030-58342-2_11. doi:10.1007/978-3-030-58342-2_11.

[17] M. Alghawazi, D. Alghazzawi, S. Alarifi, Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review, Journal of Cybersecurity and Privacy 2 (2022) 764–777. URL: https://www.mdpi.com/2624-800X/2/4/39. doi:10.3390/jcp2040039.

[18] P. Vats, A. Saha, An overview of sql injection attacks, MatSciRN: Other Materials Performance (Topic) (2019). doi:10.2139/ssrn.3479001.

[19] J. Majumder, G. Saha, Analysis of sql injection attack (2012). doi:10.47893/ijcsi.2013.1102.

[20] H. Shahriar, S. North, W.-C. Chen, Early detection of sql injection attacks, International Journal of Network Security & Its Applications 5 (2013) 53–65. doi:10.5121/IJNSA.2013.5404.

[21] S. Roy, A. Singh, A. Sairam, Detecting and defeating sql injection attacks, International Journal of Information Engineering and Electronic Business (2011). doi:10.7763/IJIEE.2011.V1.6.

[22] A. Kiezun, P. J. Guo, K. Jayaraman, M. D. Ernst, Automatic creation of sql injection and cross-site scripting attacks, in: 2009 IEEE 31st International Conference on Software Engineering, 2009, pp. 199–209. doi:10.1109/ICSE.2009.5070521.

[23] N. Singh, M. Dayal, R. Raw, S. Kumar, Sql injection: Types, methodology, attack queries and prevention (2016) 2872–2876.

[24] C. Byzdra, G. Kozieł, Analysis of the defending possibilities against sql injection attacks 13 (2019) 339–344. doi:10.35784/jcsi.1329.

[25] Z. Guo-xiang, Sql injection attacks in web application, Information Security and Communications Privacy (2010).

[26] Z. Ping, Detecting and preventing sql injection attacks in oracle, Information Security and Communications Privacy (2007).

[27] B. K. Ahuja, A. Jana, A. Swarnkar, R. Halder, On preventing sql injection attacks, in: ..., 2015. doi:10.1007/978-81-322-2650-5_4.

[28] M. Qbea'h, M. Alshraideh, K. Sabri, Detecting and preventing sql injection attacks: A formal approach, 2016 Cybersecurity and Cyberforensics Conference (CCC) (2016) 123–129. doi:10.1109/CCC.2016.26.

[29] M. Alghawazi, D. Alghazzawi, S. S. Alarifi, Deep learning architecture for detecting sql injection attacks based on rnn autoencoder model, Mathematics 11 (2023) 3286. doi:10.3390/math11153286.

[30] E. Hosam, H. Hosny, W. Ashraf, A. S. Kaseb, Sql injection detection using machine learning techniques, in: 2021 8th International Conference on Soft Computing & Machine Intelligence (ISCMI), 2021, pp. 15–20. doi:10.1109/ISCMI53840.2021.9654820.

[31] U. Bhatt, A. Xiang, S. Sharma, A. Weller, A. Taly, Y. Jia, J. Ghosh, R. Puri, J. M. F. Moura, P. Eckersley, Explainable machine learning in deployment, in: Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, 2019. doi:10.1145/3351095.3375624.

[32] T. Ha, S. Lee, S. Kim, Designing explainability of an artificial intelligence system, in: Proceedings of the Technology, Mind, and Society, 2018. doi:10.1145/3183654.3183683.

[33] T. Li, L. Han, Dealing with explainability requirements for machine learning systems, in: 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), 2023. doi:10.1109/COMPSAC57700.2023.00182.

[34] A. Bueff, I. Papantonis, A. Simkute, V. Belle, Explainability in machine learning: a pedagogical perspective, ArXiv abs/2202.10335 (2022).

[35] R. Byrne, Counterfactual thought, Annual review of psychology 67 (2016) 135–157. doi:10.1146/annurev-psych-122414-033249.

[36] K. Epstude, N. J. Roese, The functional theory of counterfactual thinking, Personality and Social Psychology Review 12 (2008) 168–192. doi:10.1177/1088868308316091.

[37] X. Dastile, T. Çelik, H. Vandierendonck, Model-agnostic counterfactual explanations in credit

scoring, IEEE Access PP (2022) 1–1. doi:10.1109/ACCESS.2022.3177783.

[38] S. S. H. Shah, SQL injection dataset | Kaggle, 2021. URL: https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset.

[39] M. Hasan, Z. Balbahaith, M. Tarique, Detection of SQL Injection Attacks: A Machine Learning Approach, 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA) (2019) 1–6. URL: https://ieeexplore.ieee.org/document/8959617/. doi:10.1109/ICECTA48151.2019.8959617.

[40] H. Gao, J. Zhu, L. Liu, J. Xu, Y. Wu, A. Liu, Detecting SQL Injection Attacks Using Grammar Pattern Recognition and Access Behavior Mining, 2019 IEEE International Conference on Energy Internet (ICEI) (2019) 493–498. URL: https://ieeexplore.ieee.org/document/8791338/. doi:10.1109/ICEI.2019.00093.

[41] Q. Li, W. Li, J. Wang, M. Cheng, A SQL Injection Detection Method Based on Adaptive Deep Forest, IEEE Access 7 (2019) 145385–145394. URL: https://ieeexplore.ieee.org/document/8854182/. doi:10.1109/ACCESS.2019.2944951.

[42] D. Tripathy, R. Gohil, T. Halabi, Detecting SQL Injection Attacks in Cloud SaaS using Machine Learning, 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS) (2020) 145–150. URL: https://ieeexplore.ieee.org/document/9123029/. doi:10.1109/BigDataSecurity-HPSC-IDS49724.2020.00035.

[43] Q. Li, F. Wang, J. Wang, W. Li, LSTM-based SQL Injection Detection Method for Intelligent Transportation System, IEEE Transactions on Vehicular Technology 68 (2019) 1–1. URL: https://ieeexplore.ieee.org/document/8616823/. doi:10.1109/TVT.2019.2893675.

[44] K. Kamtuo, C. Soomlek, Machine Learning for SQL injection prevention on server-side scripting, in: 2016 International Computer Science and Engineering Conference (ICSEC), IEEE, 2016, pp. 1–6. URL: http://ieeexplore.ieee.org/document/7859950/. doi:10.1109/ICSEC.2016.7859950.

[45] D. Das, U. Sharma, D. K. Bhattacharyya, Defeating SQL injection attack in authentication security: an experimental study, International Journal of Information Security 18 (2019) 1–22. URL: http://link.springer.com/10.1007/s10207-017-0393-x. doi:10.1007/s10207-017-0393-x.

[46] Ö. Kasim, An ensemble classification-based approach to detect attack level of SQL injections, Journal of Information Security and Applications 59 (2021) 102852. URL: https://linkinghub.elsevier.com/retrieve/pii/S2214212621000867. doi:10.1016/j.jisa.2021.102852.

[47] M. R. Ul Islam, M. S. Islam, Z. Ahmed, A. Iqbal, R. Shahriyar, Automatic Detection of NoSQL Injection Using Supervised Learning, 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC) 1 (2019) 760–769. URL: https://ieeexplore.ieee.org/document/8754304/. doi:10.1109/COMPSAC.2019.00113.

[48] N. M. Sheykhkanloo, Employing Neural Networks for the Detection of SQL Injection Attack, Proceedings of the 7th International Conference on Security of Information and Networks 2014-Septe (2014) 318–323. URL: https://dl.acm.org/doi/10.1145/2659651.2659675. doi:10.1145/2659651.2659675.

[49] M. Robnik-Sikonja, Improving random forests, in: Proceedings of the European Conference on Machine Learning, 2004, pp. 359–370. doi:10.1007/978-3-540-30115-8_34.

[50] G. Biau, Analysis of a random forests model, Journal of Machine Learning Research 13 (2010) 1063–1095. doi:10.5555/2503308.2343682.

[51] M. T. Keane, E. M. Kenny, How case-based reasoning explains neural networks: A theoretical analysis of XAI using post-hoc explanation-by-example from a survey of ANN-CBR twin-systems, in: K. Bach, C. Marling (Eds.), Case-Based Reasoning Research and Development - 27th International Conference, ICCBR 2019, Otzenhausen, Germany, September 8-12, 2019, Proceedings, volume 11680 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 155–171. URL: https://doi.org/10.1007/978-3-030-29249-2_11. doi:10.1007/978-3-030-29249-2_11.

[52] S. M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, p. 4768–4777.