IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 27, 2020, accepted January 8, 2021, date of publication January 19, 2021, date of current version January 27, 2021.

*Digital Object Identifier 10.1109/ACCESS.2021.3052956*

# Pattern Mining and Detection of Malicious SQL Queries on Anonymization Mechanism

**JIANGUO ZHENG AND XINYU SHEN**
Glorious Sun School of Business and Management, Donghua University, Shanghai 200051, China
Corresponding author: Xinyu Shen (2191050@mail.dhu.edu.cn)

**ABSTRACT** With the striking development of big data, individual privacy and data security obtain unprecedented importance. Database anonymization mechanism is created for protecting individual privacy by adding noise to the result of a query, which finds a tradeoff between the privacy and utility of personal data. However, corresponding attacks are emerging continuously resulting in a high risk of individual identification. In this paper, we learn patterns of malicious SQL queries and propose a novel detection method. Association rules are used to mine patterns and features of noise-exploitation attacks, and parse trees are applied to the feature extraction of SQL, thereby we construct feature vectors and input them into the classifiers. At the same time, we also propose a SQL generation method to generate query samples based on a real database for model training and testing. Experiments show that our detection method can significantly prevent noise-exploitation attacks including almost all differential attacks and 91% cloning attacks based on the synthetic dataset, which ensures a strong degree of data utility.

**INDEX TERMS** Database anonymization mechanism, diffix, malicious query detection, SQL parsing, random forest.

## I. INTRODUCTION

In recent years, people embrace an era of data explosion. A large amount of data is being generated, transmitted, and collected at all times, therefore the connection between individuals and data is getting intimate. While the development of science and technology has brought visible convenience to people, the problem of personal privacy leakage has arisen spontaneously. Privacy leaking events are emerging in an endless stream and are increasing gradually, among which, Facebook [4] leaked account information of 5000 users in 2017, and the anonymous movie rating database published by Netflix [2], [3] was reconstructed. It can be seen that privacy leaking issues occur all the time, highlighting the importance of individual privacy and data protection.

In order to solve this problem, database anonymization methods came into being, mainly by adding noise to data to protect individual privacy. Finding a balance between data privacy and utility, while achieving the data protection for data analysis, is the goal of current anonymization mechanisms. It also is a challenge and opportunity in the era of big data. Many traditional anonymization mechanisms can protect the privacy of individuals to a certain extent, but some limitations and risks still exist. For example, k-anonymous [1] has the risk of database reconstruction. Differential privacy [6] is a mainstream privacy protection model based on strict mathematical definitions, which is one of the most promising privacy models at present. However, in order to protect the privacy of each individual in the database, when the data volume is small, a large amount of noise should be added, which will influence the data utility. Besides, it is confronted with a series of inference attacks simultaneously.

Then, in recent years, the research enthusiasm for anonymization mechanisms has been increasing, among which Diffix [7], [8] is a representative one. It is a highly accurate database anonymization mechanism proposed at the end of 2017. Aircloak [9] is a company that provides privacy-preserving analytics for data-driven business, enabling organizations to gain flexible and secure insights into sensitive datasets mainly through Diffix and it widely offers services in the field of banking, healthcare and telecommunications. Diffix plays the role of SQL intermediary between a data analyst and an original database. The sticky noise it achieves breaks through the traditional noise adding mechanism. Although this feature can make it resist most of the attacks, such as averaging attacks. But due to the features including an unlimited number of queries and rich SQL syntax, differential

The associate editor coordinating the review of this manuscript and approving it for publication was Vicente Alarcon-Aquino.

attacks and cloning attacks can be adopted to easily obtain noise samples through a large number of queries, which will result in individual identification.

In fact, not only Diffix, but many anonymization mechanisms are at risk of being attacked, which will be introduced in detail in Section II. Therefore, this paper will take Diffix as a special case to study defense methods against its attacks, and it can be extended to most anonymization mechanisms. According to the defense method proposed by Andrea Gadotti *et al.*[5], the set of queries generated by their noise-exploitation attacks follow a specific template, and learning this pattern may help prevent this kind of attacks, as well as potentially related attacks. That is to say, if the input of a malicious query can be detected, such attacks can be prevented to a great extent.

Therefore, this research will first review the principle of anonymization mechanisms and related query attacks. Then, we take Diffix as an example to learn the patterns and characteristics of malicious queries. Finally, a novel method is proposed to detect malicious queries against anonymization mechanisms.

**Contributions.** The main contributions of this paper can be summarized as below:

- We collect and analyze the characteristics of malicious SQL query statements, which are conducive to mining the pattern of malicious query attacks by association rules.
- We decompose and extract SQL structured features by introducing the concept of parse tree, and construct feature vectors of SQL text.
- Based on the real database, we generate both malicious and normal SQL samples as the synthetic datasets for experimental analysis.
- The essence of malicious query detection problem is a classification problem. According to the feature vectors of SQL, we used random forest classifier to perform detection and achieved an accuracy over 94%, which can prevent almost all differential attacks and 91% cloning attacks based on our synthetic dataset. Our method can be extended to most anonymization mechanisms.

**Paper organization.** The rest of the paper is organized as follows. Related works and background are summarized in Section II. The working principle of Diffix anonymization mechanism is briefly described in Section III. In Section IV, we compare the existing popular query attacks and exploit the characteristics of malicious queries. In Section V, we explain the malicious query detection method proposed in this paper, and take Diffix as an example to show the design, implementation and application of the detection filter layer. The method of constructing and generating SQL samples is explained in Section VI. In Section VII, we do the experimental analysis, calculating and evaluating the accuracy of the detection method, and the probability of a successful noise-exploitation attack on Diffix after setting the detection filter layer. Finally, we summarize the research results of this paper and discuss further defensive measures and future work in the last section.

## II. RELATED WORK

### A. ANONYMIZATION MECHANISM AND QUERY ATTACKS

Anonymization mechanisms provide a certain guarantee for data privacy and security. But unfortunately, with the deepening of data mining and analysis, de-anonymization, re-identification and reconstruction have become easier and easier. Cynthia Dwork *et al.* stated the existing attack types in literature [11], and specifically listed attack methods such as linear statistical reconstruction, exponential attacks, attacks requiring only polynomial numbers, and noise tracking attacks. Aloni Cohen *et al.* [12] proposed a linear programming reconstruction attack in 2019 and pointed out that Diffix provided unlimited queries and was very vulnerable to linear reconstruction attacks. Andrea Gadotti *et al.* [10] successfully implemented differential attacks and cloning attacks on Diffix, and proved that related attacks were still effective. An endless stream of query attacks continues to test the security of anonymization methods.

### B. DEFENSE AGAINST QUERY ATTACKS

In present, in response to the successful query attacks proposed in the existing literature [10], [12], Aircloak has tried to make some repairs to Diffix to strengthen the defense measures against SQL query attacks. The fixing plan mainly puts forward two points. 1) Identify the attribute columns that are vulnerable to attack, and prohibit the use of mathematics in the query statement that can uniquely identify the attribute value of an individual. 2) Detect the most common 200 attribute values in each column. When the query conditions do not meet any of these values, it is judged as a dummy condition and the query is prohibited. It can be seen that Diffix mainly defends against query attacks by restricting query conditions, that is, identifying and restricting query statements of potential attackers. However, such a fixing scheme is still one-sided and cannot fully cover the characteristics of malicious queries. A more complete and systematic method of malicious query detection is needed. There are also studies that continue to optimize the noise mechanism. While, under the premise of satisfying the infinite queries and ensuring the balance of data utility and privacy, other mechanisms different from Diffix's sticky noise will inevitably fall into the situation of excessive noise and data distortion. At the same time, there is also a possibility of noise being exploited. Therefore, the only way to successfully and effectively defend against query attacks and ensure minimal noise addition is to filter and block potentially malicious queries. Andrea Gadotti *et al.* [5] also proposed that noise-exploitation attacks actually follow a specific pattern after implementing malicious query attacks. If malicious queries can be detected, such attacks can be prevented. Thus, the detection of malicious queries is particularly important, but there are few related kinds of research on it. Fortunately, the defense and detection of SQL injection attacks in Web security, another area where SQL is used as a medium to attack databases, is more mature.

### C. DETECTION OF SQL INJECTION ATTACKS IN WEB

In the field of web attacks, malicious query attacks are very common. Attackers can intrude on the server by injecting malicious code into the parameters of the HTTP requests, which is extremely harmful. Compared with few pieces of research on the detection of malicious queries on anonymization mechanisms, there have been various methods and tools for the detection of SQL injection attacks [15]–[17], [27]. Traditional detection methods mainly focus on methods based on classification, clustering, statistics and information. Buehrer *et al.* [21] proposed a detection method to verify the parse tree of SQL queries in 2005, which prevents SQL injection attacks by comparing the parse trees of SQL statements before and after user input in real-time. Ogheneovo, E *et al.* [25] constructed a parse tree model based on the SQL syntax structure to detect SQL injection queries through the results of dynamic parse trees in 2013. Due to the structural characteristics of SQL statements, the features of query statements can be extracted well based on parse trees, and this method can also be used in this research to identify and extract features of SQL statements. McWhirter *et al.* [22] extracted the features of SQL based on the gap-weighted string subsequence kernel in 2018, and used SVM for classification with an accuracy rate of over 92.48%. Many literatures used machine learning for detection [18], [24]. For example, Joshi *et al.* [27] used the naive Bayes classification method for detection in 2014. Rawat [20] *et al.* used SQL string tags to construct a feature matrix and used SVM for classification in 2012. Rong *et al.* [28] proposed a character-level CNN-based malicious network request detection method in 2018. CNN can solve the problem that the feature matrix cannot reflect the relationship between words in the previous research, and at the same time, it is aimed at unknown new types that cannot be discovered by the current detection system, and build a relearning model.

### D. COMPARISON OF SQL ATTACKS ON THE WEB AND ANONYMIZATION MECHANISMS

Although SQL is the main form in both attacks, the ideas of detecting them are not the same. Actually, the detection methods of SQL injection attacks cannot be directly transferred to the detection of malicious queries in anonymization mechanisms. SQL sentences in SQL injection attacks have distinct characteristics themselves and have a unified pattern for different types of databases. But query attacks in anonymization mechanisms are closely related to the target database. Although fixed patterns also exist, detecting this kind of query attack requires real-time interaction with the database to obtain relevant information such as sensitive attribute fields, unique personal identification attributes, common attribute values and so on. It cannot be directly judged whether it is malicious or not by relying on independent SQL query statements only. Therefore, detecting malicious queries in anonymization mechanisms is much more complicated. Luckily, SQL injection attack detection methods can still be referred to, especially in SQL statement feature recognition, feature matrix construction and classifying fields.

## III. SUMMARY OF DIFFIX FRAMEWORK

Diffix [7], [8] is a new anonymization mechanism, which assumes a SQL proxy between the data analyst and the database. Fig. 1 shows the implementation framework of Diffix. When the data analyst sends a request, Diffix will convert it and send it to the database, then the database will refer to the real query data table of Diffix. Finally, the noisy result will be returned to the data analyst after processing. The proposer claims that Diffix is an alternative to differential privacy, which can provide users with unlimited queries, unlimited SQL query semantics, and minimal noise addition.

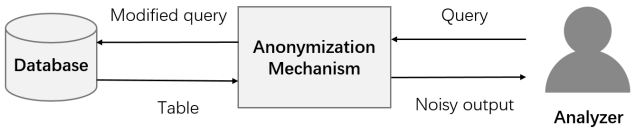


**FIGURE 1.** Framework of diffix.

Diffix breaks through the traditional noise adding mechanism. It does not directly add noise to the real result to do perturbation but uses 'sticky noise' to add noise to each query condition. It is called 'sticky' because Diffix always returns the same result for the same query.

In Equation 1, we suppose that a data analyst is sending a SQL query to Diffix. Here is a notice that 'count(*)' is written as 'count' for brevity in this paper. When the noise is generated, static noise will first be added to the gender attribute, age attribute and income attribute. For instance, the static noise seed of gender attribute can be seen in Equation 2. The difference between static noise and dynamic noise lies in the personal set of 'uid's in the dataset. Dynamic noise will be added to the set of satisfied users for each attribute. The dynamic noise seed of gender attribute is shown in Equation 3. Here, we just briefly stated the basic principles and process of Diffix's sticky noise. For more details, please refer to literature [8].

$$\begin{aligned} Q &= {}'SELECT\ count\ FROM\ A\ WHERE \\ sex &= \text{`}Female'\ and\ income = \text{`}>= 50K'' \quad (1) \\ static_{seed} &= XOR(sex, salt) \quad (2) \\ dynamic_{seed} &= XOR(static_{seed}, hash(uid_1), \\ &\quad hash(uid_2), \ldots) \quad (3) \end{aligned}$$

## IV. PATTERNS AND CHARACTERISTICS OF MALICIOUS QUERIES

### A. MAIN TYPES OF ATTACKS

Taking Diffix as an example, Table 1 shows the most mainstream attack types, methods and characteristics. According to [10], the most popular attacks are averaging attack, differential attack, cloning attack and linear reconstruction attack.

**TABLE 1. Characteristics of main types of attacks.**

| Type | Description | Feature |
|---|---|---|
| Averaging attack | Repeat one query, averaging results to approximate the true result. | Repeatability |
| Differential attack | Two query conditions differ by only one attribute, resulting in a difference in the results, particular attribute value for an individual can be inferred. | Identifying attributes |
| Cloning attack | Multiple query statements with the same syntax and different semantics with dummy conditions are issued to obtain noise samples for individual inference. | Dummy conditions |
| Linear reconstruction attack | Multiple sets of query statements with non-intersecting query ranges and dummy conditions are issued to obtain noise samples for individual inference. | Dummy conditions and grouping queries |

#### 1) AVERAGING ATTACK

Including differential privacy and other anonymization methods achieved by adding noise, the attacker can realize the averaging attack by submitting multiple repeated queries [13]. Under the condition that the number of queries are not limited and the sample size obtained is large enough, the attacker can be infinitely close to the correct result by calculating the average of the noise-added results. For instance, the attacker sends a SQL statement to query Alice's income and repeats the same query to obtain Alice's real income. Therefore, in order to avoid such averaging attacks, Diffix uses sticky noise to deal with it. Since Diffix does not directly add noise to the result of the query, it adds dynamic or static noise to each attribute involved in the query condition. Under averaging attacks, the result obtained by the static noise perturbation cannot make the attacker directly and accurately infer whether it is close to the real result. So, the existing Diffix mechanism can resist the averaging attack well.

#### 2) DIFFERENTIAL ATTACK

By submitting multiple sets of two query statements with a containment relationship in one set, the difference of the results is obtained to infer the specific attribute value of a certain individual. For instance, the query target in statement 1 is all the specific conditions, and the query target in statement 2 is the result of excluding an individual under the condition of sentence 1. To illustrate with a simple example, we can refer to Equation 4 and Equation 4. What we need to get is whether Alice's income exceeds 50,000. Without adding noise, the attacker can directly send this type of query combination to the database to get the result. Under Diffix, because it has made its noise mechanism public, the added noise satisfies the Gaussian distribution. Through this prior knowledge, the attacker constructs relevant differential query combinations, analyzes the probability distribution of the returned results, and can obtain the target with a high probability of the attribute value of the individual. In literature [10], Algorithm 1 shows the process of the differential attack. $R_1$ represents the samples of the target attribute value 0 and $R_2$ represents the samples of the target attribute value 1. $f_1$ and $f_2$ are proved to be the probability distributions of the samples in the two corresponding cases. By respectively bringing $R_1$ and $R_2$ into these two distributions for the likelihood ratio test

**Algorithm 1** Differential Attack

**Input:**
known attributes and value A,$x_A$, unique attributes and value U, $x_U$, target attribute s

**Output:**
True when $x_s = 1$, False when $x_s = 0$

1: **for** i←1 to n **do**
2: $Q_1 \leftarrow$ 'SELECT count FROM T WHERE $A = x_A$ AND $x_s = 0$'
3: $Q_1^* \leftarrow$ 'SELECT count FROM T WHERE $A = x_A$ AND $x_s = 0$ AND U$\neq x_U$'
4: **if** $Q_1 > 0$ and $Q_1^* > 0$ **then**
5: $q_{1i} \leftarrow Q_1 - Q_1^*$
6: $R_1 \leftarrow R_1 \cup \{q_{1i}\}$
7: **end if**
8: **end for**
9: **for** i←1 to n **do**
10: $Q_2 \leftarrow$ 'SELECT count FROM T WHERE $A = x_A$ AND $x_s = 1$'
11: $Q_2^* \leftarrow$ 'SELECT count FROM T WHERE $A = x_A$ AND $x_s = 1$ AND U$\neq x_U$'
12: **if** $Q_2 > 0$ and $Q_2^* > 0$ **then**
13: $q_{2i} \leftarrow Q_2 - Q_2^*$
14: $R_2 \leftarrow R_2 \cup \{q_{2i}\}$
15: **end if**
16: **end for**
17: $f_1 \leftarrow$PDF of $N(0,2)$, $f_2 \leftarrow$PDF of $N(1,2n+2)$
18: L$\leftarrow \prod_{q_1 \in R_1} \frac{f_1(q_1)}{f_2(q_1)} \prod_{q_2 \in R_2} \frac{f_1(q_2)}{f_2(q_2)}$
19: **return** L $\geq$ 1

in step 18, the target attribute value can be inferred.

$$
\begin{aligned}
Q_1 &= 'SELECT\ count\ FROM\ T \\
&\quad WHERE\ income =' <= 50k'' \\
Q_1^* &= 'SELECT\ count\ FROM\ T \\
&\quad WHERE\ name \neq 'Alice'\ AND\ income =' <= 50k'' \\
Q_2 &= 'SELECT\ count\ FROM\ T \\
&\quad WHERE\ income =' > 50k'' \qquad (4) \\
Q_2^* &= 'SELECT\ count\ FROM\ T \\
&\quad WHERE\ name \neq 'Alice'\ AND\ income =' > 50k'' \\
& \qquad (5)
\end{aligned}
$$

**Algorithm 2** Cloning Attack

**Input:**
known attributes and value A,$x_A$, unique attributes and value U, $x_U$, dummy conditions $D_1$,$D_2$,...,$D_n$, empirical variance $\sigma$

**Output:**
True when $x_s = 1$, False when $x_s = 0$

1: **for** i←1 to n **do**
2: $\beta \leftarrow A = x_A$ AND $D_i$
3: $Q \leftarrow$ 'SELECT count FROM T WHERE $\beta$ AND $x_s = 1$'
4: $Q^* \leftarrow$ 'SELECT count FROM T WHERE $\beta$ AND $x_s = 1$ AND U $\neq x_U$'
5: $q_i \leftarrow Q - Q^*$
6: **end for**
7: $\overline{q} \leftarrow \frac{1}{n}\sum_{i=1}^{n} q_i$, $S_q^2 \leftarrow \frac{1}{n-1}\sum_{i}^{n}(q_i - \overline{q})^2$
8: **return** $S_q^2 \geq \sigma$

#### 3) CLONING ATTACK

Such attacks follow the same syntax but different semantics. For Diffix, this attack is very deadly. Since Diffix adds noise based on the attributes of the query, when the syntax of the SQL query is the same but the semantics are different, the attacker can obtain a large number of noise samples for analysis. There is a high risk of individual identification. In cloning attacks, the actual query targets of the query statements in one set are the same. While due to the noise mechanism of Diffix, the results will be different. The attacker can obtain the true result by analyzing the distribution of noise and the probability distribution of the returned results. In the example of a query attack shown in Equation 6, the attacker has prior knowledge of Alice in the database and her age. By sending multiple sets of queries similar to Equation 6, a large number of noise samples about Alice's income can be obtained because 'age $< 24$' and 'age $< 24.5$' are actually the same in terms of the real query results. Yet under the Diffix mechanism, different noise addition results will be returned. Based on the literature [10], Algorithm 2 simply shows the process of related cloning attacks. $q_i$ is the sample obtained by the attacker, and the average value of all samples is $\overline{q}$, then the sample variance $S_q^2$ can be computed in step 7. Finally, the target attribute value is determined to be 0 or 1 based on the comparison between the sample variance $S_q^2$ and the empirical variance $\sigma$.

$$\begin{aligned} Q &= 'SELECT\ income\ FROM\ T\ WHERE \\ name &= 'Alice'\ and\ age > 22\ AND\ age < 24' \\ Q^* &= 'SELECT\ income\ FROM\ T\ WHERE \\ name &= 'Alice'\ and\ age > 22\ AND\ age < 24.5' \end{aligned} \quad (6)$$

#### 4) LINEAR RECONSTRUCTION ATTACK

This kind of attack solves the value of the real result by constructing a target plan to minimize the error as the objective function and assumes that the error satisfies the Gaussian distribution. As shown in Equation 7, it is a typical linear reconstruction attack, which performs linear reconstruction by grouping the unique identification attribute 'uid's for the individual. Since the characteristics of this attack combine which of the differential attack and cloning attack together, we do not emphasize too much on it. Also in literature [12], more details of this attack can be referred to.

$$\begin{aligned} Q_1 &= 'SELECT\ count\ FROM\ T\ WHERE \\ uid &> 1000\ AND\ uid < 2000\ and \\ floor(uid) &= floor(uid + 0.5)\ AND\ income =' <= 50k'' \\ Q_2 &= 'SELECT\ income\ FROM\ T\ WHERE \\ uid &> 2000\ AND\ uid < 3000\ and \\ floor(uid) &= floor(uid + 0.5)\ AND\ income =' <= 50k'' \end{aligned} \quad (7)$$

### B. CHARACTERISTICS OF MALICIOUS SQL QUERIES

Generally speaking, malicious queries may appear in pairs or groups, which include both normal queries and malicious queries with possible correlations. To avoid being recognized, some sophisticated attackers will not directly submit the entire set of malicious queries. Under this circumstance, the goal of detection is to determine whether it is malicious or not should be based on a single SQL query statement. In fact, only detecting a single malicious query submitted by a current attacker is able to destroy the structure of the entire query attack. Whether there is a normal query or a correlation in the potential query group, the target result of the malicious query is missing, then the probability of a successful attack will be greatly reduced. Taking the differential attack as an example in Equation 4 and Equation 4. After the input $Q_1^*$ and $Q_2^*$ are identified as malicious queries, the attacker will not be able to acquire useful samples to analyze the results, thus protecting Alice's range of income.

According to the types and instances of current main attacks demonstrated, some characteristics of malicious SQL statements can be obtained, and one usually has one or more of these characteristics. For example, based on the analysis of the main attack types, malicious SQL query statements will always contain sensitive attributes or dummy conditions.

Sensitive attributes refer to attributes that may make individuals at risk of being identified. Under the condition of certain background knowledge, an attacker may try to use sensitive attributes to obtain target information of a specific individual by means of differential attacks. Among these attacks, the unique personal identification attribute is particularly important. In the target dataset, this attribute can uniquely identify individual data such as name, student number and so on. Therefore, sensitive attributes should be considered to identify SQL characteristics.

For dummy conditions, it is more difficult to detect them than searching for sensitive attributes only. There are many forms and characteristics of dummy conditions. Sophisticated attackers will constantly change and construct various dummy conditions to obtain more noise samples for

exploitation to obtain real results. Fortunately, there are still rules to follow. Through the malicious SQL statements listed above, the characteristics of the dummy conditions in the query conditions are as follows: (1) The attribute value in the condition does not meet the data type. (2) The condition contains sensitive mathematical functions. (3) The condition contains abnormal an attribute value.

### C. PATTERN MINING

We can find that the summarized characteristics may be correlated and dependent in a malicious query. When a malicious query has feature A, there is a great possibility that feature B also exists. Then, the most suitable method to describe this kind of pattern is the association rule mining method. Therefore, we choose association rule mining to conduct pattern mining of malicious queries.

Based on the analysis of characteristics of a malicious query, we manually select five feature items for pattern mining. Table 2 shows the detailed description.

**TABLE 2. Description of manually selected feature items.**

| Feature item | Description |
|---|---|
| A | including sensitive attributes |
| B | meeting data type |
| C | including abnormal attribute value |
| D | including identifying attributes |
| E | including sensitive math function |

According to the association rule mining [26], we set the minimum support to 0.05 and the minimum confidence to 0.6. Among them, the support is the frequency of the feature set in the malicious query set,

$$sup = P(X) = \frac{|\{t \in T, X \in t\}|}{T}. \tag{8}$$

The confidence is defined as the probability that a sentence contains both feature set X and feature set Y for the rule X $\Rightarrow$ Y,

$$conf(X \Rightarrow Y) = P(X \cup Y|X) = \frac{sup(X \cup Y)}{sup(X)}. \tag{9}$$

Apriori approach and FP-tree approach are both used to find frequent itemsets and the potential association rules. According to Apriori [30], Algorithm 3 is demonstrated. FP-tree is an improvement based on Apriori without generating candidate sets to reduce time complexity. Part of the results of the association rules between the various features of malicious queries can be seen in Table 3 and Table 4, then the basic patterns of query attacks can be obtained. We can find that there is no significant difference in the running time of the two methods in this paper.

In order to well understand the result of the association rule, we can take a deep sight in Table 3. For example, $\{E, B\} \Rightarrow \{D\}$ means that when there is an existence of identifying attributes, the probability that there will appear

**Algorithm 3** Pattern Mining of Malicious SQL Queries

**Input:**
feature dataset D, minimum support $sup_{min}$, minimum confidence $conf_{min}$

**Output:**
association rules R

$C_1 \leftarrow$ getCandidate1(D)
**for** each candidate $c \in C_1$ **do**
  **if** $c_{count} \geq sup_{min}$ **then**
    $L_1 = L_1 \cup c$
  **end if**
**end for**
**for** $(k \leftarrow 2; L_{k-1} \neq \varnothing; k++)$ **do**
  $C_k \leftarrow Apriori(L_{k-1})$
  $L_k \leftarrow \{c \in C_k | c_{count} \geq sup_{min}\}$
**end for**
**for** each frequent itemset $l \in L_k$ **do**
  $S \leftarrow getAllSubset(l)$
  **for** $i \leftarrow 1$ *to* $size(S)$ **do**
    $s_1 \leftarrow S[i]$
    $s_2 \leftarrow s_2.filter(s_1)$
    $conf \leftarrow P(s_1 \cup s_2|s_1)$
    $R \leftarrow R \cup \{\{s_1 \in S\} \Rightarrow \{s_2 \in S\}|conf \geq conf_{min}\}$
  **end for**
**end for**
**return** R

**TABLE 3. Association rule based on Apriori.**

| Association rule | Confidence |
|---|---|
| {A} $\Rightarrow$ {C} | 0.72 |
| {B} $\Rightarrow$ {D} | 0.80 |
| {E} $\Rightarrow$ {C} | 0.75 |
| {E} $\Rightarrow$ {D} | 0.75 |
| {A,E} $\Rightarrow$ {C} | 0.80 |
| {E,B} $\Rightarrow$ {D} | 0.80 |
| {C,E} $\Rightarrow$ {A} | 0.67 |
| {A,B,D} $\Rightarrow$ {C} | 0.67 |
| {B,C,E} $\Rightarrow$ {A} | 0.67 |
| {A,D,E} $\Rightarrow$ {C} | 0.67 |

**TABLE 4. Association rule baesd on FP-tree.**

| Association rule | Confidence |
|---|---|
| {B,E} $\Rightarrow$ {D} | 0.80 |
| {A,E} $\Rightarrow$ {C} | 0.80 |
| {C,E} $\Rightarrow$ {A} | 0.67 |

a sensitive math function or a wrong data type of an attribute is 0.80.

The use of association rule mining on the characteristics of query sentences proves that malicious queries are traceable, and also confirms the viewpoint of literature [10]. Therefore, the detection of malicious queries can be further realized through the classification of SQL sentence feature vectors.

## V. DETECTION MODEL

### A. MODEL DESIGN

This section mainly introduces the malicious query detection model for the anonymization mechanism. In the application of anonymization mechanism, the detection model will be set before the original anonymization mechanism as the detection filter layer, as shown in Fig. 2. After the data analyst submits the query request, the detection filter layer sends the processed SQL statement to the database, and after obtaining the relevant data, it classifies the target query SQL statement. If it is a normal query, the query will be returned to the data analyst with noisy results through the anonymization mechanism; if it is malicious, the query will be prohibited. Next, we will focus on the implementation of the detection model to detect the query submitted by the data analyst.

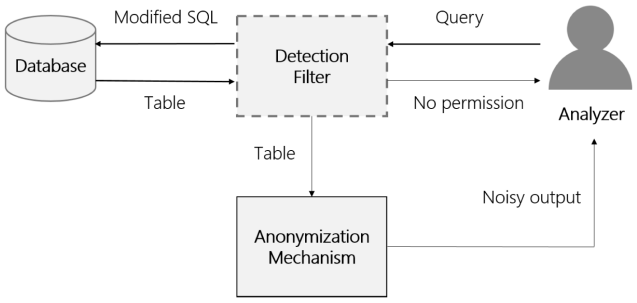


**FIGURE 2.** Detection filter in the framework.

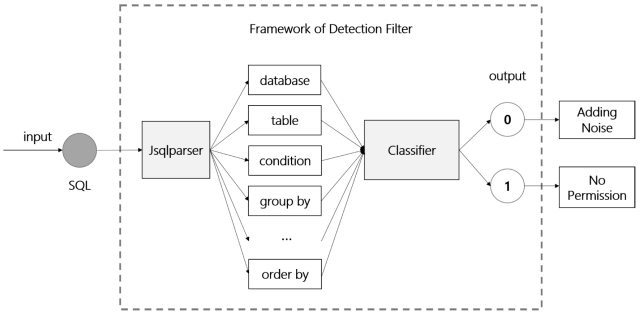


**FIGURE 3.** Design of detection filter.

Fig. 3 shows the design of the specific detection model. When the data analyst enters the SQL statement, the parse tree will be used to preprocess the SQL statement first, and extract the database name (database), table name (table), condition (where), grouping (group by), sorting (order by) and other ingredients. Subsequently, the processed SQL feature data enters the classifier to generate a feature matrix based on the mined query attack features, and then classify and output through the trained classifier, then finally get the query result whether it is malicious, where 0 represents normal queries and 1 represents malicious queries.

### B. EXTRACTING SQL COMPONENTS

SQL is a structured query language. Based on the principle of compilation, the relevant elements can be extracted by compiling SQL. JSqlParser is a SQL grammar interpreter. This research mainly uses JSqlParser to parse SQL statements and extract relevant query lists, query tables, and query conditions.

In the compilation process of extracting SQL sentence components, it will mainly go through the lexical analysis and grammatical analysis. The main function of lexical analysis is to recognize words in sentences. The lexical analyzer includes input buffer, preprocessing subroutine, scanner, and scanning buffer, and so on. It eliminates meaningless characters such as spaces, blanks, and newlines, and switches according to specific states. The graph recognizes meaningful strings such as words, numbers, operators, and identifiers. The task of grammatical analysis is mainly used to analyze the structure of a sentence. There are bottom-up and top-down methods for analysis. The bottom-up method starts from the input string, gradually reduces, and constructs the grammar tree from the leaf nodes, which represents the operator-first analysis method, while the top-down method starts from the beginning symbol of the sentence and gradually deduces it from the tree starting from the roots to construct a syntax tree, which represents a recursive descent analysis method.

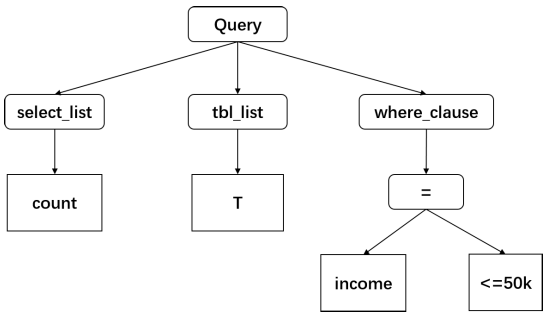


**FIGURE 4.** SQL parse tree.

In terms of the compilation of SQL statements, we can find that the parsed syntax tree in Fig. 4 is corresponding to the SQL statements in Equation 10 during lexical analysis. We apply JSqlParser for lexical analysis, and identify the SELECT, FROM, and WHERE identifiers in the sentence including the words 'count', 'A' and 'income', and the operator '=' and the character string '<=50k'. According to the grammatical characteristics of SQL, the sentence can be grammatically analyzed. Finally, the query target is 'count', the table name is 'T', and the query condition is 'income = '<=50k''.

$$Q = '\ SELECT\ count\ FROM\ T \\ WHERE\ income = '\ <= 50k'' \quad (10)$$

### C. CLASSIFIER

For the features of the extracted SQL statements, an 0-1 variable is used. When the element components of the target statement match the description of the corresponding feature, the feature value is 1, and vice versa. Based on the

characteristics and patterns of malicious queries exploited in Section IV, we redefine the six classifying features as below:

- sensitive: whether sensitive attributes are included
- type: whether data types are met
- abnormal: whether abnormal values exist in condition
- identifier: whether identification value is included
- math: whether math function is included
- label: whether the SQL is malicious

Table 5 shows the feature value and classifications of the query samples previously illustrated.

**TABLE 5. Features of SQL and classification.**

| sensitive | type | abnormal | identifier | math | label |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 |

Therefore, this problem can be summarized as a two-class classification problem of multiple features. According to the features and labels of the existing SQL samples, we classify the target SQL query sentence based on supervised learning. However, on account of there are only five features along with the two corresponding feature values (0 and 1), only 32 sets of data can be obtained at most. Such a small size of the dataset can be easy to fall into overfitting during the training process of the classifier. In order to solve this overfitting problem and increase the generalization and robustness of the model, we adopt a method of data augmentation by adding random noise to the dataset, as shown in Table 6.

**TABLE 6. Features of SQL and classification after noise.**

| sensitive | type | abnormal | identifier | math | label |
|---|---|---|---|---|---|
| 0.093 | 1.236 | 0.234 | 0.674 | 0.238 | 0 |
| 1.098 | 1.432 | 0.432 | 1.459 | 0.124 | 1 |
| 1.321 | 1.249 | 0.092 | 1.231 | 0.987 | 1 |
| 1.348 | 0.129 | 1.236 | 1.459 | 0.322 | 1 |
| 0.471 | 0.284 | 1.276 | 0.384 | 1.972 | 1 |

By this method, we generated the size of datasets of 10, 100, 1,000, 10,000, and 100,000, and take 80% as the training set and 20% as the testing set. Random forest, decision tree, naive Bayes, and logistic regression were used for model training. We set the maximum depth of the tree to 3 in the random forest and decision tree to prevent them from overfitting. Fig. 5 shows the training results under different sizes of the dataset and different classifiers.

Fig. 5 and Fig. 6 show the accuracy of the four classifiers for malicious query classification. It can be seen that as the training data set increases, the overall indicators presented an upward trend. The results of the overall index showed that the performance of Naive Bayesian classification was the worst. Its accuracy was around 0.75, and its f1-score

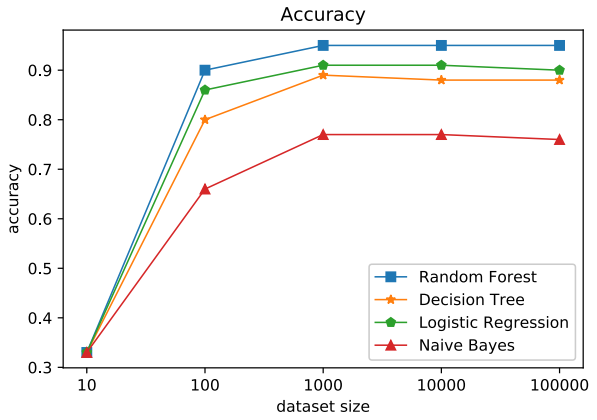


**FIGURE 5. Accuracy of classifiers.**

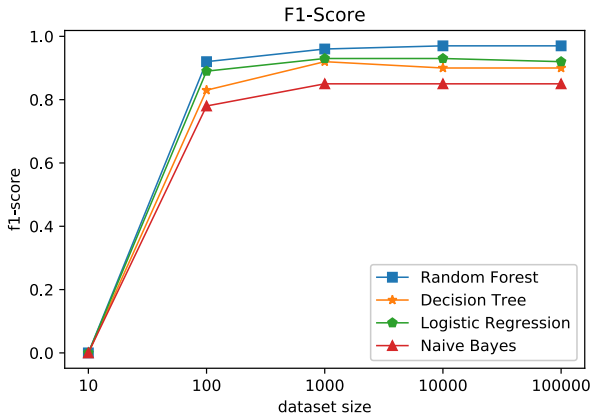


**FIGURE 6. F1-score of classifiers.**

was also the lowest, eventually converging to about 0.85. While its recall finally approached 0.99, which indicates that the Naive Bayesian classification will not miss any possible malicious query even if it misjudges a normal query. This feature protects the security of the data to the greatest extent, but this feature will affect the utility of the data greatly.

The performance of both decision tree and logistic regression was at a good level. The accuracy of decision tree finally converged around 0.88, and its f1-score was 0.90. The accuracy of logistic regression finally converged 0.90, meanwhile, its f1-score was 0.93. From this point of view, logistic regression is slightly better than decision tree.

The principle of random forest classifier is multiple decision trees, which can avoid overfitting of a single decision tree. Its accuracy was as high as 0.95 and its f1-score was 0.97. To determine which one of random forest and logistic regression is more suitable, we should measure and compare the misjudgment of normal queries more accurately between them.

Fig. 7 shows the f1-score for normal query classification. It can be found that the results of random forest were better than those of logistic regression. In the model testing, the final accuracy of random forest for normal queries was close to 0.90, while logistic regression was 0.84. When the data

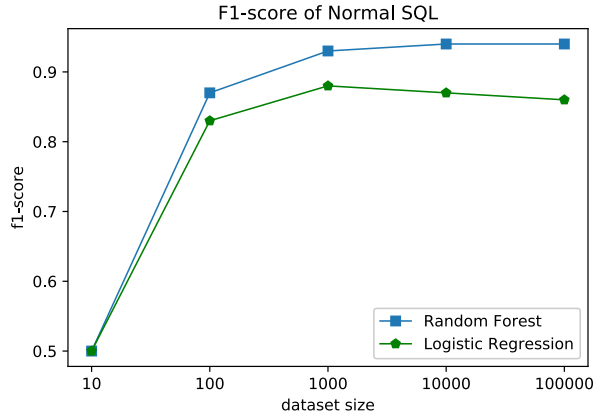


**FIGURE 7.** F1-score of normal SQL classification.

sample size is large enough, the f1-score of the random forest was close to 0.94, while the logistic regression was only 0.89. Therefore, the performance of the random forest classifier is the best among all classifiers. Besides, it can greatly avoid the misjudgment of normal queries while ensuring the accuracy of malicious query detection.

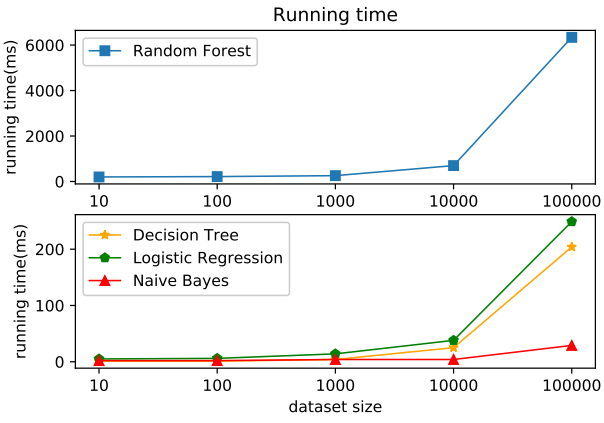


**FIGURE 8.** Training time of classifiers.

In terms of model training time, when the amount of data increased significantly, the running time required for the random forest model increased exponentially. Fig. 8 shows that, with a data volume of 100,000, the training time of random forest was 6682ms, decision tree was 174ms, logistic regression was 366ms, and Naive Bayes took the smallest time which was only 36ms. This is because the random forest is classified as ensemble learning, and when the amount of data is very large, it takes more time than the other three individual learners. Random forest uses multiple CART decision trees as weak learners. Each decision tree is independent of the other. After inputting a sample, it depends on each decision tree for judgment and classification, and the final classification result is determined by voting.

CART decision tree uses Gini coefficient as the criterion for feature selection. In Equation 11, GI is denoted as Gini coefficient, K is the number of classes, and $p_{mk}$ represents the weight of class k in node m.

$$GI_m = \sum_{k=1}^{K} \sum_{k' \neq k} p_{mk}(1 - p_{mk'}) = 1 - \sum_{k=1}^{K} p_{mk}^2. \quad (11)$$

When the Gini coefficient of an attribute is larger, it indicates that the attribute has a stronger ability to reduce sample entropy, that is, the attribute has a stronger ability to change the data from uncertainty to certainty.

Random forest uses the bagging method to randomly select features when generating decision trees. Each tree in the forest is generated by random sampling, and the sampling is replaced to ensure the randomness and unbiasedness of each tree. For random forest classification, only one parameter needs to be determined, that is, the number of decision trees to be generated. Generally speaking, the more the number of decision trees, the higher the accuracy of its training, but with it the increase in computational overhead.

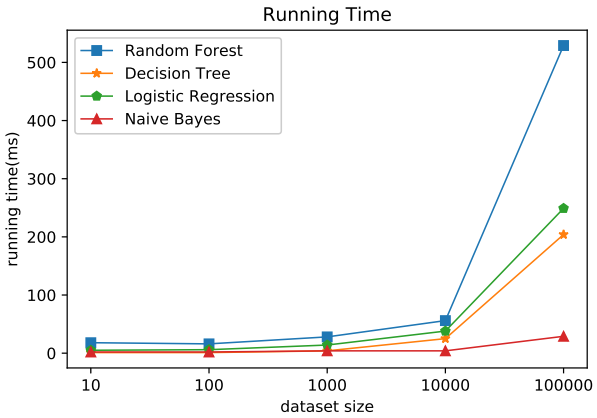


**FIGURE 9.** Improved training time of classifiers.

Therefore, it is proposed to optimize the training time of the random forest model by re-adjusting the number of decision trees. According to the number of features and samples of the query classification to set the parameter of $n_{estimators}$, we reset the number of decision trees of 5. Fig. 9 shows the training time of each classifier after adjusting the parameters. It can be seen that the training time of the random forest was significantly reduced, the time was only 529ms with a data volume of 100,000, and the accuracy was 0.92, and the indicators were still the best among all classifiers.

After comparing the accuracy, recall, f1-score, and training time of the model, the random forest classifier is selected for the classifier of malicious query detection in this paper. At the same time, it is able to calculate the variable importance measures(VIM) of attributes in the classification based on the Gini coefficient. First, it calculates the Gini index $GI_m$ (see Equation 11), and then calculate the importance of feature $X_j$ at node m, that is, the change in Gini index before and after the branch of node m (see Equation 12),

$$VIM_{jm}^{(Gini)} = GI_m - GI_l - GI_r \quad (12)$$

Among them, m is the number of features, and $GI_l$ and $GI_r$ represent the Gini index of the two new nodes after branching. If the node of feature $X_j$ appearing in decision tree i is in the set M, then the importance of $X_j$ in the $i^{th}$ tree is

$$VIM_{ij}^{(Gini)} = \sum_{m \in M} VIM_{jm}^{(Gini)} \tag{13}$$

Assume that N is the number of trees in the forest, then the importance of feature $X_j$ in all trees is

$$VIM_{j}^{(Gini)} = \sum_{i=1}^{N} VIM_{ij}^{(Gini)} \tag{14}$$

After normalizing all the obtained variable importance scores, the importance of feature $X_j$ is obtained as $VIM_j$,

$$VIM_j = \frac{VIM_j}{\sum_{i=1}^{m} VIM_j} \tag{15}$$

In Table 7, the top three characteristic attributes are sensitive attributes, personal identification attributes, and abnormal attribute values. Combined with the results of the association rules, it again confirmed that malicious query attacks follow a certain pattern.

**TABLE 7. Importance of features.**

| Feature | Importance |
|---|---|
| sensitive | 0.411 |
| type | 0.071 |
| abnormal | 0.126 |
| identifier | 0.378 |
| math | 0.012 |

## VI. COLLECTION AND PREPROCESSING OF SQL

Since there is currently no existing malicious query detection dataset for anonymization mechanism, data in this study will be collected and structured mainly through the following two methods. (1) Collect information about such malicious queries in SQL statement samples of existing research work (2) Generate SQL query statements based on the structured characteristics of SQL.

For the second method, on the premise that the SQL syntax is correct, the SELECT query statement is constructed in combination with the database of the target SQL statement. The main syntax is as follows:

SELECT select_list
FROM table_source
[ WHERE search_condition ]
[ GROUP BY group_by_expression ]
[ HAVING search_condition ]
[ ORDER BY order_expression [ ASC | DESC] ]

Briefly speaking, it can be divided into three steps to generate SQL statements. (1) Obtain target data table information. (2) Randomly select target objects. (3) Combine SQL components to generate statements. Fig. 10 shows the main process.

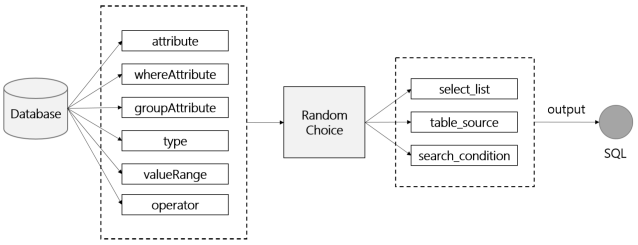


**FIGURE 10. Process of constructing SQL samples.**

**TABLE 8. Description of variables.**

| Variable | Full name | Description |
|---|---|---|
| A | attribute | array of attributes |
| WA | whereAttribute | array of attributes in condition sentence |
| GA | groupAttribute | array of attributes in grouping sentence |
| T | type | map between attribute and its corresponding datatype |
| OP | operator | array of operators |
| VR | valueRange | map between attribute and its value range |

The information such as the target field, the corresponding data type, the range of attribute values, and common enumeration classification values can be obtained through interaction with the target database. After obtaining the field array, data type dictionary, and other information, the target element can be randomly obtained by using the random selection function on the dictionary or array of the target element. Take the selection query target (select_list) as an example, one of the elements in the field array can be randomly selected as the query target by using the random selection function.

When it comes to setting query conditions, the number of conditions needs to be determined. Then the selection of attributes, operators, and attribute values is the same as which of the previous one. According to the current query field, we randomly select and set the condition field, operator, and attribute value in the corresponding array. Among them, most of the operator arrays are operators such as'>','<','=' and so on. The distribution of attribute values is either normal or not meeting the uniform distribution to ensure the balance of the dataset. Attribute values can be a certain value, or it can include the use of mathematical functions.

After selecting all the elements, identifiers such as SELECT, FROM, and WHERE can be combined according to the grammatical rules of the SQL query statement to generate the final SQL.

Based on the Adult database, Table 9 partially shows some of the results of SQL samples generated by this method.

## VII. EXPERIMENTAL EVALUATION

We conducted an experimental analysis and evaluation based on the detection method proposed in this article by using synthetic datasets generated by the method in Section VI. The constructed SQL sample dataset was mainly based on the Adult database [29] and Credit database from Kaggle.

**Algorithm 4** SQL Generation

**Input:**
attribute A, whereAttribute WA, operator OP, valueRange VR, number of conditions N

**Output:**
SQL sample

```
getTableInfo()
select_list ← random(A)
table_source ← T
condition ← ''
for i←1 to N do
    a ← random(WA)
    o ← random(OP)
    v ← random(VR)
    c ← a + o + v
    condition ← condition + c
end for
SQL ← "SELECT" + select_list + "FROM" + table_source + "WHERE" + condition
return SQL
```

**TABLE 9.** Partially generated SQL samples.

| SQL | Label |
|---|---|
| SELECT count FROM adult WHERE age > 40 | 0 |
| SELECT count FROM adult WHERE income = '>=50k' | 0 |
| SELECT income FROM adult WHERE uid = 100 | 1 |
| SELECT count FROM adult WHERE sex = Female | 0 |

All fields can be obtained by interacting with the corresponding table of the database, and sensitive attributes and other information can be set according to the actual situation. JSqlParser was introduced to compile the SQL statement and extract the components needed for feature recognition. Then, we compared each SQL query with the database to make a judgment and constructed feature vectors. Finally, the trained random forest classifier was applied to query classification.

### A. EVALUATION OF DETECTION MODEL

First, we generated two sets of samples based on dataset Adult and Credit respectively, and then classified them based on our model. Fig. 11 shows that the final classification accuracy was 0.96 based on the Adult and the accuracy of the Credit reached 0.94. It has proved that our approach is applicable.

Secondly, the two main types of Diffix attacks are tested and evaluated. For differential attacks, the detection accuracy in both sets of samples reached 1.0. Due to the fact that the characteristics of SQL sentences in differential attacks are very distinct, the conditions will always include attributes that can uniquely identify individuals. What's more, there is no need for redundant dummy conditions. Therefore, our model can detect differential attacks with great confidence. For cloning attacks, the detection accuracy was 0.91 in Adult and 0.88 in Credit. Although the characteristics of malicious queries in cloning attacks are more diverse and complicated, the final classification accuracy still achieved a good result according to the exploited patterns.

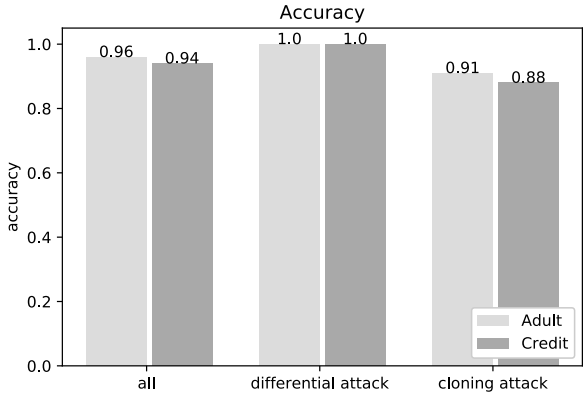


**FIGURE 11.** Accuracy of malicious query classification.

**TABLE 10.** Performance of model comparison.

| Model | Average Accuracy |
|---|---|
| This work | 0.95 |
| TextCNN | 0.81 |

In order to confirm the effectiveness of the proposed feature extraction method, we compared our model with TextCNN, which is a classic text classification method. It is also the basis of detection methods in SQL injection attacks [28]. Collected and generated training data was utilized to train the model, and the same synthetic datasets were tested. Finally, the average accuracy of the two models in Table 10 was obtained. The average accuracy of our work was 0.95, while TextCNN was around 0.81. It can be seen that the result of our model was greater than TextCNN. This indicates that it is not enough to detect malicious queries of anonymization mechanisms only by the characteristics of the independent SQL statements themselves. The interaction with the target dataset is needed. Therefore, the exploited patterns and feature extraction in this paper are effective and applicable.

### B. PRIVACY GUARANTEE

We carried out a particular noise-exploitation attack to test privacy guarantee of the improved Diffix anonymization mechanism after applying the detection filter layer to it.

Aiming at the differential attack, the literature [10] used the ratio test of the difference between results to determine its distribution, deciding the value of a certain attribute of the target attribute. Under our improved model, the attacker can get almost all the results of normal queries. But due to that our proposed model can detect almost all the differential attack, there is no suitable noise sample returned for further analysis. Therefore, compared with the differential attack in the literature [10], in which, an individual can be identified with an

**TABLE 11. Precision between malicious and normal queries.**

| Query | Precision |
|---|---|
| malicious query | 0.94 |
| normal query | 0.89 |

accuracy of 0.894 using only 5 attributes, our improved Diffix anonymization mechanism can nearly resist all differential attacks based on the synthetic dataset.

For cloning attacks, the attacker compares the variance of the noise sample results with the empirical variance to determine the target value. Literature [10] used cloning attacks to achieve an accuracy of 0.933 with only 32 queries per user. In this experiment, the filtering layer can detect nearly all the dummy conditions with the identifying attributes of an individual based on the synthetic dataset. Therefore, the absence of such noise samples totally breaks the logic of the original cloning attack algorithm, which makes the attack lose its efficacy.

It has been proved that our proposed detecting method has a strong privacy guarantee. In the meantime, it is not only effective for Diffix, it can also be extended to more anonymization mechanisms. In fact, although there are endless kinds of query attacks, they will all follow a specific pattern, like which exploited in this paper. Therefore, in terms of the anonymization mechanism of interactive queries, our detection model can be considered as the basis to defend against potential query attacks.

### C. PRIVACY-UTILITY TRADEOFF

In data analysis, the balance between data privacy and data utility is a key issue. By calculating the interception rate of malicious queries and the passing rate of normal queries, we evaluated Diffix after adding the detecting filter based on the synthetic dataset of Adult. Here, the interception of queries is used as the criterion, and the impact of Diffix's method of adding noise to data on the utility and privacy is not involved.

According to the results of detection and classification, the final precision for malicious query classification was 0.94, and for normal query classification was 0.89. It can be seen that after adding the detection and filtering layer, the data privacy of Diffix can be improved. At the same time, the passing rate of normal queries can be ensured, which ensures a high degree of data utility.

## VIII. CONCLUSION

This paper proposes a novel method of malicious query detection to resist noise-exploitation attacks against anonymization mechanisms. A filtering layer is added to the original framework to directly block suspicious and malicious query statements. So as to greatly reduce the possibility of personal identification, we apply the association rule mining approach to exploit attacking patterns. Parse trees are utilized to extract and identify features of SQL statements to construct feature vectors. In terms of classification, four machine learning classifiers are mainly chosen for model training and testing, in the meantime, random noise is added to the training samples to prevent overfitting. Among all the classifiers, the random forest classifier performs best, finally reaching an accuracy of 0.91.

Due to the lack of a current database of SQL queries, we provide a SQL generation method to generate query samples based on a real database for the experiments. According to the synthetic dataset of Adult, the accuracy of identifying malicious queries is 0.96. Among them, the accuracy of identifying differential attacks almost reaches 1.0, and which of cloning attack is 0.91. What's more, an accuracy of 0.89 is achieved for the classification of normal queries, which indicates that our detection model is reliable and high in utility.

However, the patterns of malicious SQL queries we exploit in this paper can only be directly applied to the existing noise-exploitation attacks and the related attacks of anonymization mechanisms. Endless new types of attacks may appear, and some new characteristics will emerge. From this point of view, the detection method is somewhat limited. But our method can quickly exploit new characteristics of attacks, and update the current model, which is extensible to other application scenarios. In the future, our work will mainly focus on the re-learning of malicious query classification to realize automatic and rapid identification and detection of new attacking patterns. Besides, the application of our detection method to differential privacy can also be considered.

## ACKNOWLEDGMENT

The authors would like to thank the editors and reviewers for taking time out from their busy schedule to review this manuscript, and are looking forward to receiving valuable feedback to improve our work further.

## REFERENCES

[1] L. Sweeney, "K-anonymity: A model for protecting privacy," *Int. J. Uncertainty, Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, Oct. 2002.

[2] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2008, pp. 111–125.

[3] A. Narayanan and V. Shmatikov, "How to break anonymity of the netflix prize dataset," 2006, *arXiv:cs/0610105*. [Online]. Available: https://arxiv.org/abs/cs/0610105

[4] F. González, Y. Yu, A. Figueroa, C. López, and C. Aragon, "Global reactions to the cambridge analytica scandal: A cross-language social media study," in *Proc. Companion World Wide Web Conf.*, vol. 2, May 2019, pp. 799–806.

[5] M. Ashwin, K. Daniel, G. Johannes, and V. Muthuramakrishnan, "*L*-diversity: Privacy beyond k-anonymity," *ACM Trans. Knowl. Discovery Data*, vol. 1, no. 1, pp. 1–52, 2007.

[6] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, nos. 3–4, pp. 211–407, 2014.

[7] P. Francis, S. P. Eide, and R. Munz, "Diffix: High-utility database anonymization," in *Privacy Technologies and Policy*, vol. 10518. Springer, Oct. 2017, pp. 141–158.

[8] P. Francis, S. Probst-Eide, P. Obrok, C. Berneanu, S. Juric, and R. Munz, "Diffix-birch: Extending diffix-aspen," 2018, *arXiv:1806.02075*. [Online]. Available: https://arxiv.org/abs/1806.02075
[9] *Aircloak*. Accessed: 2014. [Online]. Available: https://aircloak.com/
[10] A. Gadotti, F. Houssiau, L. Rocher, B. Livshits, and Y. A. De Montjoye, "When the signal is in the noise: Exploiting Diffix's sticky noise," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 1081–1098.
[11] C. Dwork, A. Smith, T. Steinke, and J. Ullman, "Exposed! A survey of attacks on private data," *Annu. Rev. Statist. Appl.*, vol. 4, no. 1, pp. 61–84, Mar. 2017.
[12] C. Aloni and K. Nissim, "Linear program reconstruction in practice," *J. Privacy Confidentiality*, vol. 10, no. 1, pp. 1–10, Jan. 2020.
[13] H. J. Asghar and D. Kaafar, "Averaging attacks on bounded perturbation algorithms," in *Proc. Privacy Enhancing Technol.*, vol. 2, 2019, pp. 358–378.
[14] Z. Ding, Y. Wang, G. Wang, D. Zhang, and D. Kifer, "Detecting violations of differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 475–489.
[15] A. Tajpour and S. Ibrahim, "SQL injection detection and prevention tools assessment," in *Proc. Int. Conf. Comput. Sci. Inf. Technol.*, vol. 9, Jul. 2010, pp. 518–522.
[16] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *J. Netw. Comput. Appl.*, vol. 60, pp. 19–31, Jan. 2016.
[17] Y. Dong and Y. Zhang, "Adaptively detecting malicious queries in Web attacks," Nov. 2017, *arXiv:1701.07774*. [Online]. Available: https://arxiv.org/abs/1701.07774
[18] P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, "Detection of SQL injection based on artificial neural network," *Knowl.-Based Syst.*, vol. 190, Feb. 2020, Art. no. 105528.
[19] I. Jemal, O. Cheikhrouhou, H. Hamam, and A. Mahfoudhi, "SQL injection attack detection and prevention techniques using machine learning," *Int. J. Appl. Eng. Res.*, vol. 15, no. 6, pp. 569–580, 2020.
[20] R. Rawat and S. K. Shrivastav, "SQL injection attack detection using SVM," *Int. J. Comput. Appl.*, vol. 42, no. 13, pp. 1–4, Mar. 2012.
[21] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using parse tree validation to prevent SQL injection attacks," in *Proc. 5th Int. Workshop Softw. Eng. Middleware (SEM)*, 2005, pp. 106–113.
[22] P. R. McWhirter, K. Kifayat, Q. Shi, and B. Askwith, "SQL injection attack classification through the feature extraction of SQL query strings using a gap-weighted string subsequence kernel," *J. Inf. Secur. Appl.*, vol. 40, pp. 199–216, Jun. 2018.
[23] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "Applied machine learning predictive analytics to SQL injection attack detection and prevention," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage. (IM)*, May 2017, pp. 1087–1090.
[24] W. Halfond, J. Viegas, and A. Orso, "A classification of SQL injection attacks and countermeasures," in *Proc. Int. Symp. Secure Softw. Eng. (SEEE)*, Mar. 2006.
[25] C. Science and P. Harcourt, "A parse tree model for analyzing and detecting SQL injection vulnerabilities," *West African J. Ind. Acad. Res.*, vol. 6, pp. 33–49, Apr. 2013.
[26] J. Hipp, U. Güntzer, and G. Nakhaeizadeh, "Algorithms for association rule mining—A general survey and comparison," *ACM SIGKDD Explor. Newslett.*, vol. 2, no. 1, pp. 58–64, Jun. 2000.
[27] A. Joshi and V. Geetha, "SQL injection detection using machine learning," in *Proc. Int. Conf. Control, Instrum., Commun. Comput. Technol. (ICCI-CCT)*, Jul. 2014, pp. 1111–1115.
[28] W. Rong, B. Zhang, and X. Lv, "Malicious Web request detection using character-level CNN," 2018, *arXiv:1811.08641*. [Online]. Available: http://arxiv.org/abs/1811.08641
[29] *Center for Machine Learning and Intelligent Systems. Adult Dataset*. Accessed: 1996. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/adult
[30] N. Domadiya and U. P. Rao, "Privacy preserving distributed association rule mining approach on vertically partitioned healthcare data," *Procedia Comput. Sci.*, vol. 148, pp. 303–312, Jan. 2019.

**JIANGUO ZHENG** received the B.S. degree in mathematics from Zhongshan University, the M.S. degree in system engineering from Xidian University, China, and the Ph.D. degree from the National Key Laboratory for Radar Signal Processing, Xidian University, in 2002. He was a Visiting Professor with the University of Ballarat, Ballarat, VIC, Australia. He is currently a Professor with the School of Business and Management, Donghua University, China. His research interests include data mining, intelligence decision systems, neural networks, techno-economic analysis, and operations management.

**XINYU SHEN** received the B.S. degree in management from Donghua University, where she is currently pursuing the master's degree in management science and engineering. Her research interests include data mining, secure data sharing, and information security.