A Bayesian model for anomaly detection in SQL databases for security systems

Citation for published version (APA):

DOI:
10.1109/SSCI.2016.7849905

Document status and date:
Published: 09/02/2017

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 31. Oct. 2023
A Bayesian model for anomaly detection in SQL databases for security systems

Madalina M. Drugan
Mathematics and Computer Science Department, Technical University of Eindhoven,
E-mail: m.m.drugan@tue.nl

Abstract—We focus on automatic anomaly detection in SQL databases for security systems. Many logs of database systems, here the Townhall database, contain detailed information about users, like the SQL queries and the response of the database. A database is a list of log instances, where each log instance is a Cartesian product of feature values with an attached anomaly score. All log instances with the anomaly score in the top percentile are identified as anomalous. Our contribution is multi-folded. We define a model for anomaly detection of SQL databases that learns the structure of Bayesian networks from data. Our method for automatic feature extraction generates the maximal spanning tree to detect the strongest similarities between features. Novel anomaly scores based on the joint probability distribution of the database features and the log-likelihood of the maximal spanning tree detect both point and contextual anomalies. Multiple anomaly scores are combined within a robust anomaly analysis algorithm. We validate our method on the Townhall database showing the performance of our anomaly detection algorithm.

I. INTRODUCTION

The precise definition of anomaly detection [1] depends on the envisioned research area, i.e. machine learning, data mining, information theory, statistics, etc. In statistics, anomaly detection is referred as outlier detection, or exceptions, whereas, in information theory, the noisy data is ignored in compressing the signal. Other similar or related topics are noise or unwanted data removal, and novelty detection of previously unobserved patterns. The problem characteristics, like the data and the anomaly type, and the application domain, like fraud detection for credit cards, insurance, determine the anomaly technique used. Detecting and analysing anomalies for security systems use different methods with different data structures. For example, network anomaly detection [2] uses distance-based outlier detection [3], whereas in database systems [4], the same method is not (straightforwardly) applicable.

Security database systems. This study focuses on anomaly detection for database security systems, which is a major issue in the SQL database systems, an important part of our everyday life. Public databases, like the Townhall and medical records, have a huge number of users and query requests every hour. Note the difference between these databases; a townhall system has only a few users, corresponding to the amount of the town hall workers, but a huge number of tables and attributes corresponding to very diverse issues handled by the community. A medical database system needs to be distributed such that the home doctors can communicate their data to specialists; thus, a medical database system has a large number of users and terminals and few entries from all the users. More sensitive database systems, from banks and governments, include private information on their citizens; the maintenance and integrity of these databases are essential. Anomaly detection for security systems deals with the identification of queries with unusual behaviour, like an invalid response from the database, unauthorised requests or a large number of similar requests that slow down the system.

A. Main contributions

We consider that the anomaly detection problem is similar to classification in semi-supervised learning [1]. In fact, in [5], learning algorithms significantly outperform other specific outlier algorithms. In this paper, we make an analogy with the typical learning of the structure of the Bayesian networks from data [6], used for example by medical expert systems.

Automatic feature extraction. In a log database, each query is labelled as either "normal" or "abnormal"; thus the class variable has two values. Each query is associated with a set of features, and the anomaly score based on the frequency of each feature in the database [7]. We consider that each query is a Cartesian product of features. Defining the features, like the length of the query, the time stamp and so on, is essential in the anomaly detection process. Each feature is classified as basic or composed (aggregated from several basic features), and independent or with dependencies. By scoring the frequencies of individual features, we can detect only so-called point anomalies. A group of features detects contextual anomalies. We compute the mutual information for each pair of features from which we generate the maximal spanning tree that detects the strongest dependencies between features [8]. The maximal spanning tree maximizes the log-likelihood of the joint probability distribution of the feature set.

Multiple score anomaly analysis. Various learning and optimization techniques, i.e. frequencies or Bayesian approaches, identify different anomalous queries. Thus, multiple anomaly scores might lead to the identification of different queries. Understanding the impact of each technique is important in assessing the performance of an anomaly detection algorithm. We compare the results obtained with different anomaly scores in terms of accuracy, sensitivity and so on.

Real database logs. TownHall is a large log of queries that cumulates about $4 \cdot 10^5$ instances, where each log instance is
characterized by 21 attributes and the initial SQL query. Beside the actual queries, the log file gives detailed information that includes parameters on 1) the user identification, 2) the response of the database system, and 3) the attributes of a query, like the length and the validation flags. We label as attacks all the query instances that contain a "SELECT *" instance, thus the queries that require unauthorized access to data and instances with an invalid response from the database. All the other instances are assumed normal. The queries in TownHall are thus mixed, attacks and normal queries, where the number of attacks is about 1/10 from the entire log database. Another difficulty in analysing this log file is the mixed type of features and their interpretation: some features have a small number of values whereas others need to be discretized.

B. Outline of the paper

Section II presents the current state-of-the-art in anomaly detection of SQL database systems. Section III introduces the preliminaries. Section IV proposes a database model with multiple scores. Section V introduces a Bayesian model for anomaly detection, whereas Section VII introduces a multiple scores anomaly analysis algorithm. Section VIII explains the practical settings concerning the experimental section. Section IX presents the experimental results. Section X concludes this paper.

II. ANOMALY DETECTION AND ANALYSIS IN DATABASE SECURITY SYSTEMS: SHORT INTRODUCTION

Specific constraints apply to anomaly detection in security database systems [1], [7]: The approach should be a "white-box" optimization transparent to the security officer. The entire process of anomaly detection and analysis should be intuitive such that both the security officer and the upset user with deleted queries could understand them. This first constraint limits the range of algorithms for this task, meaning that many trendy algorithms like support vector machines (SVM) and artificial neural networks (ANN) cannot be directly used. Behaviour based approaches that learn (on-the-fly) new threats are opposite to rule-based approaches where rules are predefined. We need techniques that segregate rare and infrequent events from anomalies, since infrequent queries in the database system might not indicate an attack, thread, etc. Furthermore, the anomaly detection algorithm should consider particularities of the database, encoded, for example, as parameter or threshold variables [9].

Anomaly detection. In the frequency based approach [10], queries are modeled as a set of features. For example, the length, the IP address, the command, the username, the time stamp of the queries are only a few features included in the database logs. A composed feature includes two or more simple features. Each query has an associated anomaly score calculated as the inverse of the product of frequency's frequencies weighted by a threshold set by a human user. Thus, a query with an average length and the standard command SELECT, which is much more widespread than INSERT, is considered normal when analysing the log. The infrequent queries, i.e. with rare feature values, are considered anomalies. Statistical measures, like false positives and true negative rates, validate the performance of the anomaly detection algorithms.

Anomaly analysis [11] is the task of analysing the performance of an anomaly detection algorithm. Database systems are split into two categories. "Normal" databases contain only well-behaved queries, whereas "leaked" databases have anomalous queries, like attacks, malicious behaviour, threads. The anomaly detection algorithm should be performed on a normal database, whereas anomaly analysis assumes that a large part of the database is leaked. Usually, these two processes are kept separately because of their different purpose in the security systems. The anomaly analysis includes different scores than anomaly detection. The severity score assesses the severity of data leakage. The severity score of a database is a function that combines the severity score of each feature assigned by the security expert. The risk score is simply the product of severity and anomaly scores.

Attack analysis assumes the availability of a vast library of attacks [11]. The attacks are classified using multiple criteria. Individual queries that contain attacks are called point attacks, i.e. unauthorized access or asking for passwords. A multiple queries attack is an unusual amount of queries coming from a single IP address such that individual queries are not an attack, but as a group, they represent an attack, because of their frequency, timing or size of response.

III. PRELIMINARIES

Each query instance \( q \) is a sequence of \( m \) distinct basic features, where \( m > 0 \) is a constant. Each feature \( F_i \) is a variable with a finite number of elements. \( F_i \) is defined as the set of all possible values of \( F_i \), denoted as the feature space of \( F_i \). Let \( f_i \) be a value in \( F_i \). By definition, a basic feature cannot be decomposed in other features. Thus, \( q = (f_{i_1}^1, f_{i_2}^2, \ldots, f_{i_m}^m) \), where \( f_{i_j}^j \) is the value of \( F_i \) for query \( q \). For practical reasons, some basic features, like the SQL query and its identifier, are ignored. Some numerical features, like the length and the size of the response, are discretized.

A composed feature \( F_{ij} \) pairs basic features \( F_i \) and \( F_j \), where \( F_{ij} = F_i \times F_j \). The number of elements of \( F_{ij} \) is finite. The feature space of the composed feature \( F_{ij} \) is defined as the Cartesian product of two feature spaces, where \( F_{ij} = F_i \times F_j \). Let \( f_{ij} \) be a value in \( F_{ij} \).

Definition 1: The query space \( Q \) is the Cartesian product of the basic feature spaces, \( Q = F_1 \times F_2 \times \ldots \times F_m \).

We assume that individual feature spaces are inhomogeneous. Thus, \( F_1 \) could be a set of numeric values, whereas \( F_2 \) could be categorical, and so on.

IV. DATABASE SYSTEMS WITH MULTIPLE SCORES

In the standard interpretation of database systems [7], each database is associated with a single anomaly score. In anomaly analysis, the sensitivity score and the labelling function define
the degree of knowledge inferred from the other databases [9]. The sensitivity score uses parameters that need to be tuned by a human user.

Consider a single anomaly score [10]. A database system is defined as a tuple \((Q, AS)\), where \(AS\) is an anomaly score selected a-priori by the user, \(AS : Q \rightarrow \mathbb{R}^+\), and \(\mathbb{R}^+\) is the set of positive real numbers. Thus, each query instance \(q\) has associated an \(AS(q)\) positive anomaly score.

Our interpretation of database systems is a generalization of the previous definitions of the SQL database for security systems. We make the analogy with learning the structure of a Bayesian network from a training set using one function, and testing the same structure on a test set using another function. Let \(AS\) be a set of \(a\) anomaly scores \(AS = \{AS_1, \ldots, AS_a\}\), where \(AS_i\) is the \(i\)-th anomaly score and \(AS_i : Q \rightarrow \mathbb{R}^+\). We consider validation scores that assess the performance of the anomaly scores, i.e. general statistic scores like accuracy, true positive rates, and so on. Let \(VS\) be a set of \(v\) validation scores \(VS = \{VS_1, \ldots, VS_v\}\), where \(VS_i\) is the \(i\)-th validation score and \(VS_i : \mathbb{R}^+ \rightarrow \mathbb{R}^+\).

**Definition 2:** A database system is a tuple \((Q, AS, VS)\), where \(Q\) is the query space, \(AS\) is a set of \(a\) anomaly scores and \(VS\) is a set of \(v\) validation scores.

In the experimental section, we consider \(a = 5\) anomaly scores and \(v = 25\) validation scores for each anomaly score.

In a practical setting, only a subset of the query space is present in a database system. In fact, the purpose of the anomaly detection algorithm is to identify types of queries which do not have instances in a database. Therefore, the current database is just a subset of the huge space of all possible databases. Let \(X_i\) be a subset of values of the \(i\)-th feature in a database, where \(X_i \subseteq F_i\). For example, the database Townhall has a validation flag indicating that the SQL database was able to respond adequately to a query (or not). The validation flag should be set on true for all queries. All the queries that have this flag set on false are anomalies, and a fair anomaly detection system identifies these queries.

**V. BAYESIAN MODELS OF SQL DATABASES**

Let \(P(F)\) be the joint probability distribution over the set of variables \(F = \{F_1, \ldots, F_m\}\) in the database \(Q\). We rewrite the corresponding joint probability distribution as a product of conditional probability distributions

\[
P(F) = \prod_{i=1}^{m} P(F_i | F_1, \ldots, F_{i-1}, F_{i+1}, \ldots, F_m)
\]

If all variables are independent, then \(P(F_i) = P(F_i | F_1, \ldots, F_{i-1}, F_{i+1}, \ldots, F_m)\). Thus, \(P(F_1, \ldots, F_m) = P(F_1) \cdot \ldots \cdot P(F_m)\). Note that the exact computation of the joint probability distribution is impractical, since its complexity is the product of the cardinality of each variable.

A Bayesian network over \(F\) is a tuple \((G, \mathcal{P})\), where \(G\) a directed acyclic graph and \(\mathcal{P}\) a set of conditional probability distributions. In the digraph \(G\), each vertex models a stochastic variable from \(F\), and the arcs model the probabilistic dependence. Each variable \(F_i\) is independent of the other non-descendent in \(F\) given its parents in \(G\). Let \(p(F_i)\) be the set of parents of variable \(F_i\). A Bayesian network’s joint probability distribution is now simplified to

\[
P(F) = \prod_{i=1}^{m} P(F_i | p(F_i))
\]

**A. A generic anomaly score**

Consider the set of instances of a specific database \(Q\), and \(q\) a specific instance, as before. Let \(N(x_i^q)\) be the number of values \(x_i^q \in X_i\), and let \(N\) be the total number of queries. The empirical probability value of the \(i\)-th variable in the database \(Q\) is \(\hat{P}_Q(x_i^q) = N(x_i^q)/N\). For any query \(q\).

**Definition 3:** The generic anomaly score \(AS : Q \rightarrow \mathbb{R}^+\) is the inverse of the joint probability value for query \(q\), where \(AS(q) = 1/\hat{P}_Q(x_1^q, \ldots, x_m^q)\).

The exact computation of the generic anomaly score is impractical, simpler dependencies between features need to be considered.

Our first anomaly score considers that all basic features are independent of each other. Thus,

**Definition 4:** The naive anomaly score is defined as

\[
AS_N(q) = \prod_{i=1}^{m} \frac{N}{N(x_i^q)}
\]

Consider now the previous anomaly score [10] defined as the inverse frequency of both basic and composed features. When only basic features are used, the frequencies based anomaly score is defined as

\[
AS_F(q) = \prod_{i=1}^{m} \frac{1}{N(x_i^q)}
\]

We ignore the threshold value of each feature, which we compare with prior probabilities established by a human expert.

**B. Justification of the Bayesian model**

We have identified two reasons why the generic anomaly score from Definition 3 is better than similar approaches in literature, see Equation 4. At first, we show that the classification task of SQL queries is degraded by simultaneously using composed and basic features in the frequency anomaly score. To argue this point of view, we use the analogy with learning Bayesian networks from data: when an attribute is copied, the classification accuracy is degraded [12]. We now extrapolate to anomaly scores. Consider an anomaly score computed over a basic feature \(X_i\) and its copy in a composed feature. The following shows that the corresponding anomaly score is biased on \(X_i\).

**Proposition 1:** The frequency based anomaly score \(AS_F\) is biased when the same feature is used twice.

**Proof:** Consider two variables \(X\) and \(Y\) each with two values, \(X = \{x_1, x_2\}\) and \(Y = \{y_1, y_2\}\). The database \(Q\) has four type of query instances: \(q_{11} = \{x_1, y_1\}\), \(q_{12} = \{x_2, y_2\}\),
we compute the mutual information function that indicates 
the basic features of Algorithm 1.

The pseudo-code of MaximalSpanningTree except the root, has exactly one parent. The root has no parent. When we consider the highest 5% anomaly scores, there are 4 false negatives, the score does not make any distinction between the two variables.

We now copy X into another variable X’ and we consider the basic variable X and the composed variable Y’.

The composed queries are now X\{Y’\}. The corresponding frequency anomaly scores are \(AS_F(x_1\{y_1x_1\}) = 1/5 \cdot 1/1\), \(AS_F(x_2\{y_1x_2\}) = 1/95 \cdot 1/4\) and \(AS_F(x_2\{y_2x_2\}) = 1/95\cdot 1/95\). Note that only queries that contain \(x_1\) are deemed anomalous considering 95 percentile.

\[
q_{21} = \{x_2, y_1\} \quad \text{and} \quad q_{22} = \{x_2, y_2\}. \quad \text{Let} \quad N = 100 \quad \text{and} \quad N(x_1) = 5 \quad \text{and} \quad N(y_1) = 5, \quad \text{with} \quad N(x_1, y_1) = 1, \quad N(x_1, y_2) = 4, \quad N(x_2, y_1) = 4, \quad \text{and} \quad N(x_2, y_2) = 91.
\]

By definition, \(AS_F(q_{11}) = 1/5 \cdot 1/5\), \(AS_F(q_{12}) = AS_F(q_{21}) = 1/5 \cdot 1/95\), and \(AS_F(q_{22}) = 1/95 \cdot 1/95\). The 91\% of the queries have a very low anomaly score, and the rest of the 9\% queries have the scores \(AS_F(q_{11})\) and \(AS_F(q_{12})\).

VI. AN ANOMALY DETECTION ALGORITHM

Consider a set of basic features identified with the database model from Section V. Our anomaly detection algorithm has two phases. In the first phase, see Section VI-A, we propose a method that extracts the pair of features with the highest mutual information, from which we construct the corresponding maximal spanning tree. Section VI-B introduces anomaly scores that use the log-likelihood and / or the maximal log-likelihood tree.

A. Maximal spanning tree for feature extraction

In this section, we generate a maximal spanning tree from an SQL database. A tree is a digraph G where each vertex, except the root, has exactly one parent. The root has no parent. The pseudo-code of MaximalSpanningTree is presented in Algorithm 1.

Consider a database system Q, where Q ∈ Q, and the basic features of Q denoted as a set of variables X = \{X₁, ..., Xₙ\}. Between each pair of variables Xᵢ and Xⱼ, we compute the mutual information function that indicates how much information Xᵢ gives about Xⱼ:

\[
I_Q(Xᵢ, Xⱼ) = \sum_{xᵢ∈Xᵢ} \sum_{xⱼ∈Xⱼ} \hat{P}_Q(xᵢ, xⱼ) \cdot \log \left( \frac{\hat{P}_Q(xᵢ, xⱼ)}{\hat{P}_Q(xᵢ) \cdot \hat{P}_Q(xⱼ)} \right)
\]

where \(\hat{P}_Q(xᵢ)\) is the probability function computed from frequency of the variable \(xᵢ\) of the feature \(Xᵢ\). The concept of mutual information is closely related to the concept of entropy since

\[
I_Q(Xᵢ, Xⱼ) = H_Q(Xᵢ) - H_Q(Xᵢ | Xⱼ)
\]

where the conditional entropy is defined as

\[
H_Q(Xᵢ | Xⱼ) = - \sum_{xᵢ∈Xᵢ} \sum_{xⱼ∈Xⱼ} \hat{P}_Q(xᵢ, xⱼ) \cdot \log \left( \frac{\hat{P}_Q(xᵢ, xⱼ)}{\hat{P}_Q(xⱼ)} \right)
\]

and the entropy for variable \(Xᵢ\) is

\[
H_Q(Xᵢ) = - \sum_{xᵢ∈Xᵢ} \hat{P}_Q(xᵢ) \cdot \log (\hat{P}_Q(xᵢ))
\]

If two variables \(Xᵢ\) and \(Xⱼ\) are independent, then their mutual information is 0, since \(H_Q(Xᵢ | Xⱼ) = H_Q(Xᵢ)\). Otherwise, \(H_Q(Xᵢ | Xⱼ) ≤ H_Q(Xᵢ)\). If two variables are copies of each other, then the mutual information has the highest value \(I_Q(Xᵢ, Xⱼ) = H_Q(Xᵢ)\).

The first step constructs the ordered set S with all the pairs of features \(Xᵢ\) and \(Xⱼ\) in the descending order of their mutual information value. We initialize the list of edges with an empty set \(T = \emptyset\). Each iteration, we select the first edge \(XᵢXⱼ\) from \(S\), and, in the same time, we remove \(XᵢXⱼ\) from \(S\). The edge \(XᵢXⱼ\) is added to \(T\), if by adding \(XᵢXⱼ\) to the edges already in \(T\), the resulting undirected graph does not contain cycles. The algorithm stops when the list \(S\) is empty. We transform the list \(T\) in a directed tree \(G\) by selecting any variable \(Xᵢ\) as a root, thus \(Xᵢ\) has no parents, and directing all the arcs such that all the edges except \(Xᵢ\) have exactly one parent. The complexity of this algorithm is \(O(m^2 \cdot \log N)\), with m the number of features and N the number of queries in the database Q.

The log-likelihood function is maximized with the maximal spanning tree over the given set of features. Thus,

\[
LL(G | Q) = \sum_{i=1}^{m} N \cdot H_Q(Xᵢ | p_G(Xᵢ))
\]

where \(p_G(Xᵢ)\) is the set of parents of \(Xᵢ\) in the digraph \(G\).

Algorithm 1 Maximal spanning tree for feature extraction

1: procedure MAXIMALSPANNINGTREE(Q)
2: Compute the mutual information between each two features \(I_Q(Xᵢ, Xⱼ)\), \(∀Xᵢ, Xⱼ \in Q\);
3: S the ordered list of mutual information values;
4: \(T \rightarrow \emptyset\) the list of edges;
5: while S is not empty do
6: Select the first pair \((Xᵢ, Xⱼ) \in S\)
7: if T remains a tree when adding \(XᵢXⱼ\) then
8: \(XᵢXⱼ\) is added to \(T\)
9: end if
10: Delete \(XᵢXⱼ\) from \(S\)
11: end while
12: Select a root \(Xᵢ\)
13: Transform \(T\) in a directed graph \(G\)
14: return \(G\)
15: end procedure
B. The log-likelihood anomaly score

In this section, we propose three anomaly score functions, from which two are log-likelihood scores and one is the inverse of the joint probability function.

Definition 5: The log-likelihood tree anomaly score is defined as

$$AS_{LT}(G, q) = -\sum_{i=1}^{m} \sum_{x_i \in p_C(x_i)} \hat{P}_Q(x_i^q, x_i^q) \log \left( \frac{\hat{P}_Q(x_i^q, x_i^q)}{P_Q(x_i^q)} \right)$$

(3)

If $X_i$ is the root, with no parents, then $p_C(X_i) = \emptyset$ and the corresponding term is $\hat{P}_Q(x_i^q) \cdot \log \hat{P}_Q(x_i^q)$. The following anomaly score is derived from the joint probability distribution of $q$ given the maximal spanning tree.

Definition 6: The tree anomaly score is

$$AS_T(q) = 1/ \left( \prod_{i=1}^{m} \prod_{x_j \in p_C(x_i)} \hat{P}_Q(x_i^q, x_j^q) \right)$$

(4)

Note the difference between this definition and the naive anomaly score, cf Definition 4. Above, we assume that there are dependencies between features, whereas the naive approach considers all the features to be independent.

At last, if we assume the independence of the features, we have the following log-likelihood score.

Definition 7: The naive log-likelihood anomaly score is

$$AS_L(q) = -\sum_{i=1}^{m} \hat{P}_Q(x_i^q) \cdot \log \left( \hat{P}_Q(x_i^q) \right)$$

(5)

C. Justification of the log-likelihood scores

Using the log-likelihood function implies using entropy as a measure of how much information is encoded by a joint probability distribution. By means of an example, we compare the naive anomaly score $AS_N$, cf Equation 1, and the log-likelihood anomaly score $AS_L$, cf Equation 5.

Example 1: Consider twenty variables $\{X_1, \ldots, X_{20}\}$, each with only two values $X_i = \{x_i^1, x_i^2\}$, where $N(x_i^1) = 5$ and $N(x_i^2) = 95$, for all $i$. The database $Q$ has also only two types of queries $Q = \{q_1, q_2\}$, where $q_1 = \{x_1^1, \ldots, x_{20}^1\}$ and $q_2 = \{x_1^2, \ldots, x_{20}^2\}$. Thus, 5% of the queries are of type $q_1$ and the rest of 95% queries are of the type $q_2$. The frequencies anomaly scores are very small $AS_{F}(q_1) = 0.20^{20}$, whereas probability based anomaly scores are very large $AS_{N}(q_1) = 20^{20}$. For some compilers (here Java), these values are out of range and prone to computation errors. The log-likelihood anomaly score $AS_L(q_1) = 0.96$ is a real positive number of reasonable range. If compiled without error, the anomalous queries are correctly identified in all examples.

A more complicated example with more diverse type queries is presented in the experimental section, where the log-likelihood score performs the best.

D. Justification of the maximal spanning tree

We now show the advantage in capturing the dependencies between variables using the same line of reasoning like in Proposition 1.

Definition 8: A feature $X_i$ is completely defined by a feature $X_j$ iff $H_Q(X_i | X_j) < \epsilon$, where the noise level $\epsilon > 0$ is a small real number.

If two variables are copies of each other they are also completely dependent since $H_Q(X_i | X_j) = 0$. The noise level $\epsilon$ is inferred from the database or set by the user. In the following proposition, we show that, unlike the frequency anomaly score, the bias introduced by the completely dependent variables is eliminated by the maximal spanning tree.

Proposition 2: The anomaly score $AS_T$ using the maximal spanning tree $G$ generated over $Q$ is not biased by the completely dependent features of the database $Q$.

Proof. Consider three variables $X, Y$ and $X'$, where $X$ and $X'$ are completely dependent. The maximal spanning tree always contains the vertex $X$ and $X'$ for which the mutual information is maximal. The joint probability distribution ignores $X'$ since $H_P(X' | X) < \epsilon$. Thus, for each query $q$, we have $AS_T(q) = \left( \hat{P}_Q(x^q) \cdot \hat{P}_Q(y^q) \right)^{-1}$, where $X$ and $Y$ are assumed independent. Therefore, the anomaly scores are independent of $X'$, and thus not biased by this dependence relation. $\square$

In the sequel, the above proposition also holds for the log-likelihood tree anomaly score $AS_{LT}$. We have $AS_{LT}(q) = \hat{P}_Q(x^q) \cdot \hat{P}_Q(y^q)$. The experimental section confirms the advantage of using the maximal spanning tree. We further comment on this issue in the experimental section.

VII. THE ANOMALY ANALYSIS METHODOLOGY

The robust anomaly analysis algorithm is the sequence of steps describing an anomaly analysis component. RobustAnomalyAnalysis is the corresponding pseudo-code of Algorithm 2. The input of RobustAnomalyAnalysis is the output of the detection algorithm from Section VI, cf Algorithm 1. The maximal spanning tree $G$ includes all the strongest dependencies between features. For each query $q_i \in Q$, we compute the associated anomaly scores: $NF$ is defined as the list of the frequency anomaly scores, cf Equation 2. By

Algorithm 2 Robust anomaly analysis

1: procedure ROBUSTANOMALYANALYSIS($G, Q$)
2:   for $i = 1$ to $N$ do
3:     $NF_i \leftarrow AS_F(q_i)$;
4:     $NP_i \leftarrow AS_N(q_i)$; $NE_i \leftarrow AS_L(q_i)$;
5:     $PP_i \leftarrow AS_{LT}(G, q_i)$; $PE_i \leftarrow AS_{SL}(G, q_i)$;
6:   end for
7: Compute percentile over $NF, NP, NE, PF$ and $PE$;
8: Calculate the performance measures for each set;
9: Compare the performance of anomaly scores;
10: end procedure
Definition 4, and \( NP \) is the list of the naive anomaly scores, cf Definition 4, and \( NE \) is the list of the naive log-likelihood anomaly score, cf Definition 7. \( PP \) is defined as the list of the tree anomaly score, cf Definition 6, and \( PE \) contains the log-likelihood tree anomaly scores, cf Definition 5. For each set of anomalies, we select the top percentile anomaly scores, i.e. 90 or 95 percentile.

Here, we consider that the database system has its own mechanism to assess attacks. The statistical metrics used as performance measures, aka accuracy, sensitivity and precision, compare the SQL queries identified by the system to be anomalous with the top percentile anomaly scores. Queries that are selected as anomalous by both the SQL system and the top percentile anomaly scores are the true positives, \( TP \). Queries that are selected as anomalous by the SQL system but not in the top percentile of anomaly score are the false negatives, \( FN \). Queries that are in the top percentile but not flagged as such by the system are false positives, \( FP \).

Finally, we compare the performance measures for all the anomaly scores. An anomaly score is considered better than another anomaly score iff the anomaly score outperforms the other scores in all metrics. In the experimental section, none of the anomaly scores are better than other anomaly scores. Thus, we consider all the anomaly scores to be informative and complementary.

VIII. EXPERIMENTAL SETTINGS

A first step in anomaly analysis for SQL database systems is to extract the important features that classify each query with high accuracy in normal or anomalous. Based on several log data files, we have the following classification of the log features: The SQL query is present in the log as a verification of the original query. The query identification attribute is a number attached to each query. The user identification attributes include the IP address, the host name and the user name. The time stamp of the query is not necessarily unique because a database could have a distributed implementation. The validation flags indicate whenever the query is considered an attack by the interrogated database system. Possible validation flags include the SELECT ALL type of queries that might indicate unauthorized access or wherever the SQL statement was correctly interpreted or not. The response indicators are attributes returned by the database, like response code, response size, and the number of rows returned from the database. The query characteristics include the type of command, like SELECT, INSERT or REMOVE, and the length of the query. The database characteristics consider the schema name (groups of databases). The data structures accessed during a query are the list of databases and attributes.

Any log database (we know) has at least the first four kinds of features. The Townhall database has all these features. The log of different databases contain different information and text formatting. Therefore, we adapt a log parser for each database. As a general observation, a variety of features makes the anomaly analysis more reliable via the interpretation of the results, but the analysis is more complicated and prone to mistakes.

A. Townhall database

To test the proposed methodology, we have used the log of the Townhall database system. We have counted 372788 valid instances in the database. Table I shows the complete set of 22 features of this database log and some of their characteristics like the number of values and their mean entropy.

For example, the user identification attribute is represented in five basic features: (5)\(^1\) the connection IP address of the host, (6) the connection hostname, (7) the host connection port, (8) the role of the database user, (9) the username. We have considered only the queries that parse correctly, meaning that they have 22 attributes, some of these attributes having a format check, i.e. numerical values, time stamps or non-empty values. Some queries have empty fields, not necessarily the same in all queries. Our analysis ignores the first two attributes that are unique for each query: (1) the SQL query, and (2) the unique identifier of each SQL. Thus, the anomaly detection and analysis algorithms consider 20 attributes in the TownHall database. The time stamps are measured in minutes, and the duration of a query, for most of the queries, is under a minute. There are, on average, 10 attributes per database instance, but some databases have much more attributes than other databases. Three features have continuous values and they are therefore discretized: (11) the size of the response, (12) the number of rows in the response, (16) the length of a query. Four attributes are either null or a list of attributes or tables: (13) list of attributes after the SELECT command, (14) list of tables after the FROM command, (19) list of attributes after the WHERE command, and (20) list of tables after WHERE command. For the INSERT command, the first two lists are empty.

B. Type of anomalies in the townhall system

SQL injections and privilege misuse are only two examples of attacks that can be initiated in a database. Only one privilege misuse is signalled in the Townhall database system by (4) the flag for "SELECT *" type of queries. However, this flag is not linked with a syntactic parser, and thus different calls of the same command SELECT ALL, are not appropriately flagged. In the sequel, SQL injections are signalled in the database by (3) the flag signalling SQL queries of unknown type.

According to the standard frequency based approach, sets of malicious queries could have the right frequencies to be considered normal, and thus the standard algorithm fails. Attacks involving groups of queries can be identified with the maximal spanning tree if variables are dependent. For example, the type of command "Unknown" coincides with the invalid flagged SQL queries, meaning that the two variables should be correlated in the maximal spanning tree.

\(^1\)The number of a feature variable in Table I
IX. EXPERIMENTAL RESULTS

In this section, we describe and discuss the experimental results of the tested algorithms and methods. Table I shows statistical measurements on the individual anomaly score sets. Besides the mean and standard deviation for anomaly scores, we have computed the top 90 and 95 percentiles anomaly scores. We consider the following performance measures. 1) Accuracy $Acc = (TP + TN)/N$, where $N$ is the total number of queries. 2) Sensitivity or true positive rate is $TPR = TP/(TP + FN)$, and 3) specificity or true negative rate is $TNR = TN/(TN + FP)$. 4) Precision or positive predictive value is $PPV = TP/(TP + FP)$. 5) F1 score is sensitivity and precision $F1 = 2 * TP/(2 * TP + FP + FN)$. 6) $F$ score combines precision and sensitivity $F = 2 * PPV * TNR/(PPV + TNR)$.

We analyse the impact of the maximal spanning tree on the performance of the detection algorithm. The structure of the maximal spanning tree generated from the Townhall database given the 20 features is simple. We have observed strong dependencies between the features describing the list of attributes and the rest of the features explicable by the large number of attributes and databases. All the features are connected with the attribute feature from the database, position (19) in Table I, which has the largest number of values implying the largest mutual information values. Since we construct a tree, all other dependencies between variables are ignored.

For a second experiment, when building the maximal spanning tree, we ignore the features with the largest number of values, naming (13), (14), (19) and (20). These four features that describe the attributes and the databases used in SQL queries, have the largest mutual information in the unpruned tree. The pruned tree includes the four vertexes but with no children and no parents when computing the corresponding $AS_{LT}$ anomaly score. As a result, some of the performance indicators of the pruned tree improved considerably whereas others decreased significantly.

In our last experiment, we decreased the percentile of the anomaly score to 90. The most robust detection algorithms vary the least with the change of percentile.

Discussion. It is interesting to note that qua anomaly scores, no score dominates all the other scores, and the difference between scores is rather small. The frequencies $AS_F$ and the naive $AS_N$ scores behave exactly the same in terms of statistical performance measures, because the two anomaly scores differ by a constant given by the size of the database. The pruned tree log-likelihood score anomaly score $AS_{LT}$ has the highest accuracy $Acc$, sensitivity $TPR$, specificity $TNR$, and $F1$ score. The second best for the above scores is the naive log-likelihood $AS_{NT}$. In opposition, the (unpruned) tree log-likelihood score $AS_{LT}$ has the largest precision $PPV$ and $F$ score, but the lowest accuracy, specificity $TNR$ and $F1$ score. When compared with the naive probability score $AS_N$, the tree based probability score $AS_T$ has a slightly larger precision $PPV$ and $F$ score but decreases in all other metrics.

As a general note for the 95 percentile experiment, the accuracy, sensitivity and $F1$ scores are quite high, whereas the specificity, precision and $F$ scores are quite low. High accuracy means a large overlap between the queries classified as anomalous by the detection algorithm, with a high anom-
lous score, and by the database system, meaning that at least one validation flag negates its normal state. Low specificity, on the other hand, shows that the false positives are quite large compared to \( T/N \). Precision shows that the number of true positive identifications is much lower than the number of false positives.

The second part of Table II shows results with a cut-off on anomaly scores at 90 percent. The ranking qua anomaly scores and performance metrics is preserved. The accuracy and \( F \) metrics are the same for all anomaly scores, except the tree based detection algorithms with an increased accuracy and \( F \) value. Both specificity and sensitivity scores increase across all the anomaly scores, but the precision and \( F \) score, which is the combination between precision and sensitivity, decreases.

Comparing now the two parts of Table II, we note a clear trade-off between precision and sensitivity. Thus, a detection algorithm with high percentile cuts-off true positive queries, whereas a lower percentile means more false positive queries.

**X. CONCLUSION**

In this paper, we propose a model of SQL databases that uses several anomaly scores. We use the analogy between the anomaly detection and learning the structure of a Bayesian network from data to generate a maximal spanning tree that captures the strongest dependencies between the features. The corresponding anomaly scores use the dependencies in the maximal spanning tree. Our anomaly analysis algorithm uses a set of other metrics, like the accuracy of the classification, to score the algorithm’ classification performance. We justify our design choices with theoretical findings.

We compare our method with the simplistic frequency based approach showing that log-likelihood methods outperform the other anomaly scores. None of the anomaly scores, however, dominates the other anomaly scores in all performance metrics. We argue that for efficient anomaly detection in database security systems, advanced optimization and machine learning tools, like cross-validation, feature selection, are necessary preserving at the same time its "white-box" paradigm. Here, we use the analogy with Bayesian networks, but more work is required to extensively apply this method. Further work is required to understand the advantages and disadvantages of the proposed method.

**ACKNOWLEDGMENTS**

M.M. Drugan is funded from ITEA M2Mgrid project.

**REFERENCES**


