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

Madalina M. Drugan
Mathematics and Computer Science Department, Technical University of Eindhoven
Email: 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 term 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

978-1-5090-4240-1/16/$31.00 ©2016 IEEE

characterized by 21 attributes and the initial SQL query. Beside the actual queries, the log datafile 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. Section VI presents an anomaly detection algorithm, 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 associated an anomaly score calculated as the inverse of the product of feature'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. $\mathbb{F}_i$ 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 $\mathbb{F}_i$. By definition, a *basic* feature cannot be decomposed in other features. Thus, $q = \{f_1^q, f_2^q, \ldots, f_m^q\}$, where f_i^q 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 $F_{ij} = F_iF_j$. 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 $\mathbb{F}_{ij} = \mathbb{F}_i \times \mathbb{F}_j$. Let f_{ij} be a value in $\mathbb{F}_{ij}$.

Definition 1: The *query space* $\mathbb{Q}$ is the Cartesian product of the basic feature spaces, $\mathbb{Q} = \mathbb{F}_1 \times \mathbb{F}_2 \times \ldots \times \mathbb{F}_m$.

We assume that individual feature spaces are inhomogeneous. Thus, $\mathbb{F}_1$ could be a set of numeric values, whereas $\mathbb{F}_2$ could be categorial, 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 $(\mathbb{Q}, AS)$, where AS is an anomaly score selected a-priori by the user, $AS : \mathbb{Q} \to \mathbb{R}^+$, 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 $AS_i : \mathbb{Q} \to \mathbb{R}^+$. 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 $VS_i : \mathbb{R}^+ \to \mathbb{R}^+$.

Definition 2: A database system is a tuple $(\mathbb{Q}, AS, VS)$, where $\mathbb{Q}$ 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 5 anomaly scores and 5 validation scores for each anomaly score, $a = 5$ and $v = 25$.

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 $\mathbb{X}_i$ be a subset of values of the i-th feature in a database, where $\mathbb{X}_i \subseteq \mathbb{F}_i$. 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 $\mathbb{Q}$. We rewrite the corresponding joint probability distribution as a product of conditional probability distributions

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

If all variables are independent, then $P(F_i) = P(F_i \mid F_1, \ldots, F_{i-1}, F_{i+1}, \ldots, F_m)$. Thus, $P(F_1, \ldots, F_m) = P(F_1) \cdot \cdots \cdot P(F_m)$. 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 $(G, \mathbb{P})$, where G a directed acyclic graph and $\mathbb{P}$ 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

$$P(F) = \prod_{i=1}^{m} P(F_i \mid p(F_i))$$

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 $x_i^q \in \mathbb{X}_i$, 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 $AS : \mathbb{Q} \to \mathbb{R}^+$ 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 the basic features are independent of each other. Thus,

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

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

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

$$AS_F(q) = \prod_{i=1}^{m} \frac{1}{N(x_i^q)} \tag{2}$$

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 by 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: $q_{11} = \{x_1, y_1\}$, $q_{12} = \{x_1, y_2\}$,

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

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 $AS_F(q_{12})$. 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 YX'. The composed queries are now $X\{YX'\}$. The corresponding frequency anomaly scores are $AS_F(x_1\{y_1x_1\}) = 1/5 \cdot 1/1$, $AS_F(x_1\{y_2x_1\}) = 1/5 \cdot 1/4$, $AS_F(x_2\{y_1x_2\}) = 1/95 \cdot 1/4$ and $AS_F(x_2\{y_2x_2\}) = 1/95 \cdot 1/91$. Note that only queries that contain x_1 are deemed anomalous considering 95 percentile. □

We conclude that the anomaly scores could be biased when mixing composed and basic features. Note that our generic anomaly score, cf Definition 3, considers only basic features. The dependencies between features are captured through the conditional probability values.

Another reason to use our generic anomaly score is the machine learning methods available for the joint probability distribution function. In the next section, we propose an anomaly detection algorithm that uses an approximation of the joint probability distribution using a maximal spanning tree.

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 $Q \in \mathbb{Q}$, and the basic features of Q denoted as a set of variables $X = \{X_1, \ldots, X_m\}$. Between each pair of variables X_i and X_j, we compute the mutual information function that indicates how much information X_i gives about X_j:

$$I_Q(X_i, X_j) = \sum_{x_i \in \mathbb{X}_i} \sum_{x_j \in \mathbb{X}_j} \hat{P}_Q(x_i, x_j) \cdot \log \left(\frac{\hat{P}_Q(x_i, x_j)}{\hat{P}_Q(x_i) \cdot \hat{P}_Q(x_j)} \right)$$

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

$$I_Q(X_i, X_j) = H_Q(X_i) - H_Q(X_i \mid X_j)$$

where the conditional entropy is defined as

$$H_Q(X_i \mid X_j) = - \sum_{x_i \in \mathbb{X}_i} \sum_{x_j \in X_j} \hat{P}_Q(x_i, x_j) \cdot \log \left(\frac{\hat{P}_Q(x_i, x_j)}{\hat{P}_Q(x_j)} \right)$$

and the entropy for variable X_i is

$$H_Q(X_i) = - \sum_{x_i \in \mathbb{X}_i} \hat{P}_Q(x_i) \cdot \log \left(\hat{P}_Q(x_i) \right)$$

If two variables X_i and X_j are independent, then their mutual information is 0, since $H_Q(X_i \mid X_j) = H_Q(X_i)$. Otherwise, $H_Q(X_i \mid X_j) \leq H_Q(X_i)$. If two variables are copies of each other, then the mutual information has the highest value $I_Q(X_i, X_j) = H_Q(X_i)$.

The first step constructs the ordered set S with all the pairs of features X_i and X_j 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 X_iX_j from S, and, in the same time, we remove X_iX_j from S. The edge X_iX_j is added to T, if by adding X_iX_j 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 X_i as a root, thus X_i has no parents, and directing all the arcs such that all the edges except X_i 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,

$$LL(G \mid Q) = \sum_{i=1}^{m} N \cdot H_Q(X_i \mid p_G(X_i))$$

where $p_G(X_i)$ is the set of parents of X_i 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(X_i, X_j), ∀X_i, X_j ∈ Q;
3:     S the ordered list of mutual information values;
4:     T → ∅ the list of edges;
5:     while S is not empty do
6:         Select the first pair (X_i, X_j) ∈ S
7:         if T remains a tree when adding X_iX_j then
8:             X_iX_j is added to T
9:         end if
10:        Delete X_iX_j from S
11:    end while
12:    Select a root X_i
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

$$AS_{LT}(G,q) = -\sum_{i=1}^{m} \sum_{X_j \in p_G(X_i)} \hat{P}_Q(x_i^q, x_j^q) \cdot \log \left(\frac{\hat{P}_Q(x_i^q, x_j^q)}{\hat{P}_Q(x_j^q)} \right) \tag{3}$$

If X_i is the root, with no parents, then $p_G(X_i) = \emptyset$ 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 the joint probability distribution of q given the maximal spanning tree

Definition 6: The *tree anomaly score* is

$$AS_T(q) = 1/ \left(\prod_{i=1}^{m} \prod_{X_j \in p_G(X_i)} \hat{P}_Q(x_i^q \mid x_j^q) \right) \tag{4}$$

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

$$AS_L(q) = -\sum_{i=1}^{m} \hat{P}_Q(x_i^q) \cdot \log \left(\hat{P}_Q(x_i^q) \right) \tag{5}$$

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 $AS_N(g_1) = 20^{20}$. 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.

Algorithm 2 Robust anomaly analysis

```
1: procedure RobustAnomalyAnalysis(G, Q)
2:     for i = 1 to N do
3:         NF_i ← AS_F(q_i);
4:         NP_i ← AS_N(q_i); NE_i ← AS_L(q_i);
5:         PP_i ← AS_T(G, q_i); PE_i ← AS_LT(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
```

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 $H_Q(X_i \mid X_j) \leq \epsilon$, 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 $H_Q(X_i \mid X_j) = 0$. The noise level ϵ 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 $H_P(X' \mid X) \leq \epsilon$. 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. □

In the sequel, the above proposition also holds for the log-likelihood tree anomaly score AS_{LS}. We have $AS_{LT}(q) = \hat{P}_Q(x^q) \cdot \log \hat{P}_Q(x^q) + \hat{P}_Q(y^q) \cdot \log \hat{P}_Q(y^q)$.

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

definition, 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

TABLE I
THE CHARACTERISTICS OF THE TOWNHALL DATABASE

	Feature	Meta Feat	type	verified	empty	nr values	entropy	remarks
1	SQL	query	string	yes	no	372788		
2	id		numeric	yes	no	372788		the unique identifier
3	valid		boolean	yes	no	2	0.0	13 queries are not valid
4	select *		boolean	yes	no	2	0.419	14.8% are "SELECT *" queries
5	user IP address	user	IP add	yes	no	3	0.716	51.9% and 47.7% for two IP addreses
6	host		string	no	yes	6	1.159	
7	port		numeric	no	yes	2	0.493	80% not specified
8	role		string	no	yes	1	0.0	Not specified
9	name		string	no	yes	3	0.513	80% unknown, 0.4% monitor
10	code	response	numeric	yes	no	4	0.228	94% a valid code
11	size		numeric	yes	no	3	2.242	discretized
12	nr rows		numeric	yes	no	11	0.727	27.8% have no response
13	list of attributes		list of strings	no	yes	469	19.936	Max 51% queries per attribute, multiple attributes per query
14	list of tables		list of strings	no	yes	9	−1.189	2291 not specified, the rest more than 2 databases per query
15	schema	database	string	no	no	2	0.006	99.9% not specified
16	sensor id		numeric	no	no	1	0	unknown functionality
17	length		numeric	yes	yes	27	1.590	discretized
18	command		string	no	no	4	0.001	"SELECT" 99.9%, "INSERT", "Update", "Unknown"
19	list of attributes		string	no	yes	955	42.893	Max 47% queries per attribute, multiple attributes per query
20	list of tables		string	no	yes	216	7.207	Max 52.9% queries per attribute, multiple attributes per query
21	start	time	date(min)	yes	no	2		discretized
22	end		date(min)	yes	no	10	0.013	99.8% queries ending in less than a minute

IX. EXPERIMENTAL RESULTS

In this section, we describe and discuss the experimental results of the tested algorithms and methods. Table II 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 significatively.

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 anoma-

TABLE II
STATISTICAL MEASUREMENTS ON THE INDIVIDUAL ANOMALY SCORES

Type anomaly type		Average score $\pm$ Std	95 **percentile**	Statistical measures %					
				Acc	TPR	TNR	PPV	F1	F
Frequencies	Frequencies AS_F	1.83E-93 $\pm$ 2.79E-181	2.86E-102	83.8	86.3	12.4	15.2	91.0	25.84
Probabilities	Naive AS_N	4.93E18 $\pm$ 2.01E42	7.7E9	83.8	86.3	12.4	15.2	91.0	25.84
	Tree AS_T	5.22E153 $\pm\infty$	3.84E108	82.9	85.9	9.2	15.7	90	26.54
Log-likelihoods	LL Naive AS_{LN}	1.97 $\pm$ 0.28	2.67	87.4	88.2	24.4	13.1	93.0	22.81
	LL Tree AS_{LT}	1.60 $\pm$ 1.91	4.64	80.2	84.4	0.0	**17.4**	89	**28.85**
	LL pruned Tree AS_{LT}	2.64 $\pm$ 0.08	3.05	**89.9**	**89.3**	**31.3**	11.9	**94.3**	21.14
Type anomaly type		Average score $\pm$ Std	90 **percentile**	Statistical measures %					
				Acc	TPR	TNR	PPV	F1	F
Frequencies	Frequencies AS_F	1.83E-93 $\pm$ 2.79E-181	4.81E-103	83.9	88.3	29.3	**12.2**	90.8	21.43
Probabilities	Naive AS_N	4.93E18 $\pm$ 2.02	1.29E9	83.9	88.3	29.3	12.2	90.8	21.43
	tree AS_T	5.69E17 $\pm$ 5.27E39	4683559	78.87	85.59	12.42	15.21	87.94	25.81
Log-likelihoods	LL Naive AS_{LN}	1.96$\pm$0.28	2.53	87.39	90.3	41.19	10.21	92.80	18.34
	LL tree AS_{LT}	1.98 $\pm$ 1.98	3.87	77.20	84.67	6.78	**16.19**	86.99	**27.18**
	LL pruned tree AS_{LT}	2.50 $\pm$ 0.09	2.98	**94.28**	**93.71**	**61.41**	6.70	**96.75**	12.5

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 TN. 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 percentile. The ranking qua anomaly scores and performance metrics is preserved. The accuracy and $F1$ metrics are the same for all anomaly scores, except the tree based detection algorithms with an increased accuracy and $F1$ 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

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, 2009.

[2] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19 – 31, 2016.

[3] E. M. Knorr, R. T. Ng, and V. Tucakov, "Distance-based outliers: Algorithms and applications," *VLDB J.*, vol. 8, no. 3-4, pp. 237–253, 2000.

[4] S. Hashemi, Y. Yang, D. Zabihzadeh, and M. R. Kangavari, "Detecting intrusion transactions in databases using data item dependencies and anomaly analysis," *Expert Systems*, vol. 25, no. 5, pp. 460–473, 2008.

[5] M. R. Smith and T. Martinez, "Improving classification accuracy by identifying and removing instances that should be misclassified," in *Proc of International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2011.

[6] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian Network Classifiers," *Machine Learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[7] E. Costante, J. den Hartog, M. Petkovic, S. Etalle, and M. Pechenizkiy, "Hunting the unknown - white-box database leakage detection," in *Proc of Data and Applications Security and Privacy IFIP*, 2014, pp. 243–259.

[8] C. K. Chow and C. N. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, pp. 462–467, 1968.

[9] S. Vavilis, M. Petkovic, and N. Zannone, "A severity-based quantification of data leakages in database systems," *Journal of Computer Security*, vol. 24, no. 3, pp. 321–345, 2016.

[10] E. Costante, S. Vavilis, S. Etalle, J. den Hartog, M. Petkovic, and N. Zannone, "Database anomalous activities - detection and quantification," in *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography*, 2013, pp. 603–608.

[11] S. Vavilis, A. I. Egner, M. Petkovic, and N. Zannone, "An anomaly analysis framework for database systems," *Computers & Security*, vol. 53, pp. 156–173, 2015.

[12] P. Langley and S. Sage, "Induction of Selective Bayesian Classifiers," in *UAI*, 1994, pp. 399–406.