DBSAFE—An Anomaly Detection System to Protect Databases From Exfiltration Attempts

Asmaa Sallam, Elisa Bertino, *Fellow, IEEE*, Syed Rafiul Hussain,
David Landers, R. Michael Lefler, and Donald Steiner

***Abstract*—Attempts by insiders to exfiltrate data have become a severe threat to the enterprise. Conventional data security techniques, such as access control and encryption, must be augmented with techniques to detect anomalies in data access that may indicate exfiltration attempts. In this paper, we present the design and evaluation of DBSAFE, a system to detect, alert on, and respond to anomalies in database access designed specifically for relational database management systems (DBMS). The system automatically builds and maintains profiles of normal user and application behavior, based on their interaction with the monitored database during a training phase. The system then uses these profiles to detect anomalous behavior that deviates from normality. Once an anomaly is detected, the system uses predetermined policies guiding automated and/or human response to the anomaly. The DBSAFE architecture does not impose any restrictions on the type of the monitored DBMS. Evaluation results indicate that the proposed techniques are indeed effective in detecting anomalies.**

***Index Terms*—Data engineering, data systems, information security.**

I. Introduction

DATA represent an extremely important asset for an organization. Exfiltration of confidential data such as military secrets, sensitive healthcare or financial data, or intellectual property poses one of the most severe threats in the case of insider cyber-attacks. A malicious insider who has the proper credentials to access the organization's databases may, over time, send data outside the organizational network through a variety of channels, such as e-mail or crafted HTTP requests that encapsulate data. Existing security tools primarily focus on protecting the organization from outside attacks. Network-level intrusion detection systems (IDS) monitor traffic patterns and attempt to infer anomalous behavior. While such tools may be effective in protecting against external attacks, they are less suitable when insiders, who have the proper credentials to access data, are exfiltrating data.

The problem of securing databases from insider threats is challenging, and its solution requires combining different techniques [1]. One relevant technique is anomaly detection (AD) by which the patterns of interaction between subjects (users or applications) and the database are analyzed to detect anomalous activity and changes in the access patterns that are indicative of early signs of exfiltration. Since much of the critical data to be safeguarded in government and corporate data stores is currently held in relational databases, we focus on securing the contents of a relational database management system (RDBMS). Our view is that an AD system that works at the RDBMS layer (i.e., at the data source) is a promising approach toward detecting data exfiltration by malicious insiders for the following reasons: 1) RDBMS access is performed through a standard, unique language (SQL) with well-understood and well-documented semantics. It is, therefore, feasible to baseline behavior at this layer, as opposed to doing so at the network or operating system layer, where the diversity of mechanisms and protocols for data transfer creates complexity that often confuses conventional IDS. 2) Monitoring the potential disclosure of confidential data is more effective when done as closely as possible to the data source. Therefore, the DBMS layer is the most suitable place for detecting early signs of data exfiltration. 3) The DBMS layer already has a thorough mechanism in place for enforcing access control based on subject credentials. Additional information about the subject requesting the data, such as role, IP address, etc., is instrumental in detecting early signs of exfiltration.

An AD system for monitoring database access needs to address several challenges: 1) The system must be able to monitor different commercial RDBMSs and integrate with logging tools provided as part of commercially available security information and event management (SIEM) systems. 2) It must have good run-time performance in order to minimize the impact on query processing times. 3) It must be able to monitor different types of data access, e.g., from users, application programs, and internal database maintenance, and variations in data access patterns, including changes in the amount of retrieved data.

In this paper, we present the design and evaluation of DBSAFE, a system to detect, alert on, and respond to anomalies in database access designed specifically for RDBMS that addresses the above challenges. A key feature of DBSAFE is

Manuscript received November 30, 2014; revised April 10, 2015, June 18, 2015, and August 23, 2015; accepted September 26, 2015. Date of publication October 26, 2015; date of current version June 26, 2017. This work was supported by the Department of Homeland Security (DHS) Science and Technology Directorate, Homeland Security Advanced Research Projects Agency, Cyber Security Division. The views expressed in this work are those of the authors and do not necessarily reflect the official policy or position of the Department of Homeland Security or Northrop Grumman Corporation. NGIS-DSEA-14-01335.

A. Sallam, E. Bertino, and S. R. Hussain are with the Cyber Center, Center for Education and Research in Information Assurance and Security (CERIAS), West Lafayette, IN 47907 USA, and also with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA (e-mails: asallam@purdue.edu; bertino@purdue.edu; hussain1@purdue.edu).

D. Landers, R. M. Lefler, and D. Steiner are with Northrop Grumman Corporation, Falls Church, VA 22042 USA (e-mails: david.landers@ngc.com; mike.lefler@ngc.com; donald.steiner@ngc.com).

Digital Object Identifier 10.1109/JSYST.2015.2487221

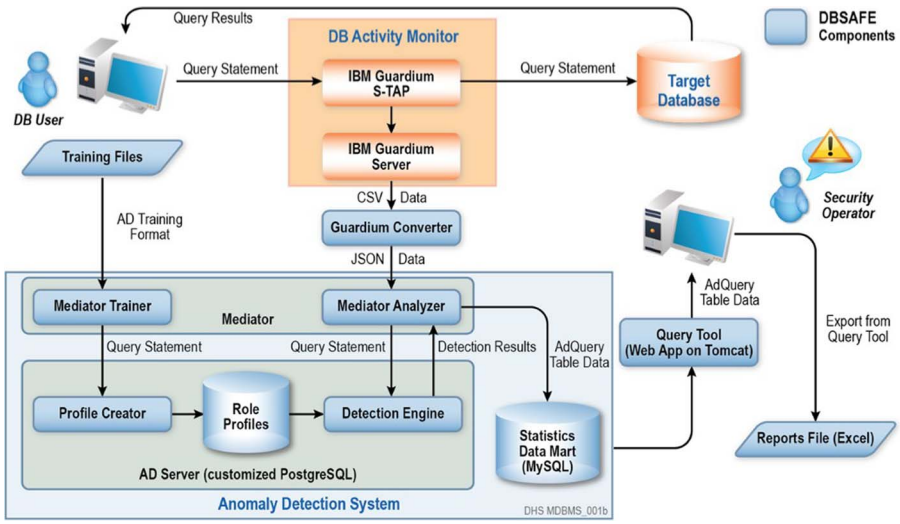

Fig. 1. DBSAFE architecture provides a scalable system for learning patterns of behavior in database interactions and detecting and alerting on deviations from these patterns.

the ability of profiling the selectivity of queries issued against the database, and to detect changes with respect to the profiled selectivity. By keeping track of the profiled selectivity, one can detect major changes in the size of data retrieved by queries. Such changes, as mentioned in [1], may be indications of data misuse. In the following, we discuss the system architecture and relevant technical issues related to creating profiles. We then report results from an extensive evaluation carried out to assess the first version of the DBSAFE prototype. We conclude by discussing related work and outlining on-going and future work.

II. DBSAFE Architecture

The key idea underlying our AD methodology is to build profiles of normal subject behavior when interacting with the monitored database during a training phase. These profiles are then used to detect anomalous behavior of these subjects. Once an anomaly is detected, response actions must be returned for possible automatic and/or human handling of the anomaly.

A first key issue in such a design is which subjects should be monitored. In our design, we assume that the database uses a role-based access control (RBAC) model. A role can be thought of as a collection of data access permissions needed to perform specific tasks. Under an RBAC model, users inherit the authorizations of the roles to which they are assigned. DBSAFE creates, manages, and monitors access pattern profiles corresponding to roles, as opposed to individual users. Using roles enables scaling the system to databases with large numbers of users since managing a few roles (and, in the case of AD, the profiles corresponding to the roles) is much more tractable than managing many individual users. Since RBAC has long been standardized in various commercial DBMS products, an RBAC-based AD solution is practical in a real world environment. In the concluding section, we outline on-going work addressing cases where roles are not used.

A second key issue is how to represent the access patterns. Very detailed access patterns may increase the accuracy of detection but, on the other hand, may be very expensive to process. To balance expressivity with efficiency, we decided to represent the access profile in terms of some query features for each query issued by each role. The query features that are extracted include: the command (e.g., SELECT, UPDATE, etc.), the tables referenced in the query (e.g., in the projection clause and the qualification clause of the query), and the columns returned by the query (e.g., the columns in the projection clause of the query). Note that this set of features characterizes the syntax of the query. In many cases, however, it is also critical to characterize the data returned by a query-anomalies in the amount of data accessed by a user represent an important indication of possible data theft. Therefore, we complement such features with an indication of the "query selectivity," that is, the estimated percentage of data retrieved from the tables in the database, as calculated by the database's query optimizer. Unlike other approaches based on inspecting the data once retrieved by the query [13], our approach makes it possible to detect anomalies in the volume of data retrieved by the query before the query is executed, thus saving system resources, enhancing performance, and reducing response times in detecting anomalies.

The DBSAFE architecture (see Fig. 1) directly supports the two main phases of our AD methodology. It comprises a profile creator component that uses audit logs as training data to create role profiles and a detection engine that inspects each query for anomalies based on these profiles. DBSAFE also includes a mediator component that integrates with different target RDBMS. As previously mentioned, access patterns are obtained by analyzing the syntactical features of a query. Such activity requires analyzing query parse trees and using the schema of the target database being monitored. The mediator extracts the required information from the target databases and imports it into an internal database used by DBSAFE to process the queries for analysis and for storing role profiles. This internal database is currently implemented using PostgreSQL.

Finally, the query interceptor, also referred to as DB Activity Monitor, gathers the actual queries being made to the target database for inspection. We used IBM InfoSphere Guardium

Data Activity Monitor [8] for this purpose. Currently, the interceptor is passive in the sense that it does not stop the queries from being passed to the target database. We are currently extending the system to support different actions to be taken upon detecting anomalies. More details about this feature can be found in Section VIII.

III. Data Representation

In this section, we describe the features extracted from the input queries as well as their internal representation. Note that this information applies to both the training queries and the queries being analyzed during the AD phase. The format of the SQL SELECT statement is as follows:

SELECT [DISTINCT] TARGET_LIST
FROM RELATION_LIST
WHERE QUALIFICATION.

Each command is internally represented by a basic data unit containing a quadruplet of fields. For the sake of simplicity, we represent a generic quadruplet using a relation of the form $Q(c, P_R, P_A, S_R)$. The quadruplet contains detailed information of a query in a log entry. c corresponds to the command type issued by the user which can be either SELECT, INSERT, UPDATE, or DELETE. P_R corresponds to the projection relation information and is represented as a binary array whose length is the number of tables in the database. The entries in the array corresponding to the tables accessed by the query have the value 1; the others have the value 0. Entries in this array are populated by data extracted from the RELATION_LIST in the FROM clause of the parsed query. P_A contains the projection attribute information in the TARGET_LIST of the query. P_A is an array of arrays where each entry is a binary array corresponding to a table in P_R and whose length is the number of attributes of the table. The length of P_A is the number of tables in the database. If an attribute is accessed in the projection list of the query, its corresponding entry in P_A will have value 1; otherwise it will have value 0. S_R is an array of length equal to the number of tables in the target RDBMS. Each entry in S_R can have one of four values: "0" if the corresponding table is not referenced in the query, "l" if more than two-third of the table is estimated to appear in the result; "m" if more than one-third and less than two-third of the table is estimated to appear in the result; and "s" otherwise. Note that different partitionings of the value estimates can easily be adopted. All components of the quadruplet are obtained from the query parse-tree except S_R which is obtained from the query plan-tree. The query plan-tree is an internal representation generated by the DBMS query optimizer which indicates the steps into which the execution of the query is decomposed. Examples of such steps are sequential scan of a table, index-based scan of a table, and hash-based join. Additional information on the plan-tree and its usage in populating the values in S_R can be found in Section V.

As an example, consider a database schema comprising the Students (s_ID, s_deptID) and Departments (d_ID, d_name) relations. Table Students contains IDs of students in column s_ID and a foreign key to Departments.d_ID in column s_deptID. Table Departments contains information about students' departments. Each record in this table has an ID and a name stored in d_ID and d_name, respectively. Data stored in both tables are shown in Table I. The corresponding quadruplet representations of some SQL queries on this database are shown in Table II.

TABLE I
EXAMPLE DATABASE

s_ID	s_deptID
1	1
2	1
..	..
10	2
11	2
..	..
20	2

(a)

d_ID	d_name
1	CS
2	ECE
..	..
10	EE

(b)

TABLE II
QUADRUPLET CONSTRUCTION

Query	Quadruplet $Q(c, P_R, P_A, S_R)$
Q1. SELECT * FROM students WHERE s_ID <10	c = "SELECT" P_R = [1,0] P_A = [[1,1],[0,0]] S_R = ['m','0']
Q2. SELECT * FROM students, departments	c = "SELECT" P_R = [1,1] P_A = [[1,1][1,1]] S_R = ['l','l']
Q3. SELECT s_ID, d_name FROM students, departments WHERE s_deptID = d_ID	c = "SELECT" P_R = [1,1] P_A = [[1,0][0,1]] S_R = ['l','s']

Note that compared with the profile format proposed in our previous approach [10], the current profile format does not include information about the attributes referenced in the WHERE-clause of the query. It however includes information about the query selectivity, that is, the S_R component. The reason why we decided to omit the information about the attributes used in the WHERE-clause is that previous experiments had shown that this information is not useful in enhancing the accuracy of detection.

IV. Profile Creation and Anomaly Detection

During the training phase, the profile creator component of DBSAFE builds profiles for roles existing in the target RDBMS. Data for training can be gathered using two different methods. The first method is to record required training information during the normal operation of the RDBMS, i.e., while users are submitting input queries to the RDBMS and expecting to get query results. The profile creator sends each input command to the parser to get its parse-tree. It then extracts features from each parse-tree and uses them to update the profile of the role corresponding to the user who issued this command. The second method for training is to use audit log files containing the previous activity of the database users. The profile creator identifies SQL commands in the log and uses them to create the profiles.

We address the problem of detecting the deviation of a user's behavior from normal as a classification problem for which we use and evaluate two different types of classifiers: 1) a "Binary" Classifier and 2) a Naive-Bayesian Classifier (NBC). Currently, we assume that one role is activated per user. The formulation of the problem and the techniques for using the classifiers are detailed in the rest of this section.

A. Binary Classifier

Given an input query and the role of the user who issued this query, the Binary Classifier's task is to decide if such a query is in the set of expected queries corresponding to this role as based on the training data. During the AD training phase, we store information in role profiles that helps the classifier perform its job during detection. The profile for a role contains a Boolean value for every attribute in each table that indicates whether this attribute has ever appeared in the projection list of a query in the training log of users in this role. During detection, the classifier checks that each of the features of the input query discussed above has appeared in a training log corresponding to the role's profile.

B. Naive-Bayesian Classifier (NBC)

The NBC has proven to be effective in many practical applications such as text classification and medical diagnosis mainly because of the low computational cost for training and detection that results from assuming that attributes considered in classification are independent. Bayesian rules of probability can then be applied with a decision rule to perform the classification. We chose to apply the Maximum-A-Posteriori decision rule (MAP) which is most commonly used with the NBC; the MAP decision rule results in correct classification as long as the correct class is more probable than all others. In the rest of this section, we describe the general principles of the NBC and show how it can be applied to our use case.

In supervised learning, a classifier is a function f that maps input feature vectors $x \in X$ to output class labels y_i, where $i \in 1, \ldots, C$, X is the feature space, and the length of the vector x is n. Our goal is to learn f from a labeled training set of N input-output pairs. One way to solve this problem is to learn the class-conditional density $\mathrm{p}(y|x)$ for each value of y and to learn the class priors $\mathrm{p}(y)$. Bayes rule can then be applied to compute the posterior $\mathrm{p}(y|x) = \mathrm{p}(x, y)/\mathrm{p}(x)$. Since the goal is to assign a class to the new instance x, a decision rule is then applied. By applying the MAP decision rule, x is assigned to the most probable class, i.e., x is assigned to the class y that maximizes $\mathrm{p}(y|x)$. Assume that the feature vector $x = [a_1, a_2, \ldots, a_n]$; the Bayesian and MAP rules can be applied as follows based on our previous results [10]:

$$y_{\mathrm{map}} \alpha \operatorname*{argmax}_{y_j \in Y} p(y_j) \prod_i p(y_j|a_i). \tag{1}$$

The NBC directly applies to our AD framework by considering the set of roles in the system as classes and the log file quadruplets as observations. The number of attributes of the system is $|P_R.P_A^T| + |P_R.S_R^T| + 1$. For example, if the database has two tables and each table has three attributes, each table may or may not be present in the query, and each attribute may or may not be accessed in the table, so the number of possible combinations of the presence of the tables and attributes is $2 * 3$, and each table can have one level of selectivity in the query; the type of the query constitutes an additional attribute of the query. The previous equation can be rewritten as

$$p(Q|r_j) = p(r_j)p(c|r_j) \prod_{i=1}^{N} p\left(\mathcal{P}_{\mathcal{R}}[i].\mathcal{P}_{\mathcal{A}}^{\mathcal{T}}[i]|r_j\right) p\left(\mathcal{P}_{\mathcal{R}}[i].\mathcal{S}_{\mathcal{R}}^{\mathcal{T}}[i]|r_j\right) \tag{2}$$

$$r_{\mathrm{map}} = \operatorname*{argmax}_{r_j \in R} p(Q|r_j). \tag{3}$$

The probability that each role sends the input query is computed using (2). In the equation, $p(r_j)$ is the prior probability, that is, the probability that the role r_j sends a query, and is equal to the number of queries issued by r_j and recorded in the training log divided by the total number of queries in the log. The rest of (2) constitutes the posterior probability that the role r_j issues the input query. $p(c|r_j)$ is the probability that a user holding the role r_j sends a query that has the same command type as the input query. $p(\mathcal{P}_{\mathcal{R}}[i].\mathcal{P}_{\mathcal{A}}^{\mathcal{T}}[i]|r_j)$ is the probability that a user who belongs to role r_j sends a query that projects columns of table i that appear in the query. $p(\mathcal{P}_{\mathcal{R}}[i].\mathcal{S}_{\mathcal{R}}^{\mathcal{T}}[i]|r_j)$ is the probability that a similar user sends a query that accesses table i and has the same selectivity as the input query. The previous two probabilities are computed for each table in the database and combined in the computation of the posterior probability using multiplication $(\prod_{i=1}^{N})$. The output of the classifier (r_{map}) is computed using (3) and is equal to the role that has the maximum probability of sending the input query; R is the set of roles in the database. After r_{map} is identified by the NBC, it is compared to the actual role of the user. If they are identical, the query is considered normal, otherwise it is considered anomalous. The procedure for computing the role as estimated by the MAP rule is shown in Algorithm 1.

Algorithm 1: NBC Algorithm

```
function get_map_role (in_query)
  Input  : The input query
  Output: Role expected by the MAP rule
for each role whose Oid rid in database do
    role_prior_prob = calculate_role_prior(rid);
    role_likelihood = calculate_role_likelihood(rid,
    in_query);
    role_log_aposteriori = role_prior_prob +
    role_likelihood;
    if role_log_aposteriori >= max_role_prob then
        max_role_prob = role_log_aposteriori;
        expected_role = rid;
    end
end
return expected_role;
```

Since the per-table selectivity component of a query (S_R) is computed based on the optimizer's output plan of the query that is based on the statistics stored on the tables in the database, when the statistics of a table change, the per-table selectivity of the training queries that reference this table may also change and should be updated accordingly. For this reason, the mediator keeps data structures in order to be able to incrementally

execute the required updates. The mediator has a hash-table that maps the identifier of each table to the list of training queries corresponding to this table. Each query has the ID of the role of the user who submitted this query and the selectivity of the table in this query. When the mediator receives new statistics for a table, it checks these queries and sends a negative-update message to the underlying AD Engine (ADE). This message has the form $(-, r_i, t_j, s_k)$ which tells the ADE that the profile of the role whose ID is r_i should be updated by decrementing $C(t_j, s_k)$, that is, the number of queries in which the selectivity of t_j is s_k. The mediator also sends similar messages for each table in the query and adds the query to a redo-list which contains queries that will be further processed by the mediator as follows. The mediator requests the ADE to compute per-table selectivities of tables in each of the queries in the list and sends positive-update messages to the ADE as follows. If a query q_j issued by a user with role r_i references a table whose identifier is t_k and the selectivity of this table in this specific query is s_{jk}, the mediator sends the ADE a message of the form $(+, r_i, t_k, s_{jk})$. The ADE responds to this message by incrementing $C(t_k, s_{jk})$. Note that by using the positive and negative updates, the ADE does not need to update other features stored in the profiles, rather only the selectivity of tables in queries in the redo-list.

C. Manual Revision of Existing Profiles

A critical issue in using classifiers is represented by the problem of incomplete profile information after the training phase because of insufficient training data. To address such problem, DBSAFE provides an interface and a tool allowing the administrator to manually add training information. This interface is also useful for making changes in access patterns of roles after the training is complete. When an attribute in a table that has not been accessed during training appears in a query during detection, a warning concerning this attribute is recorded in the log of the database. By using the tool, the administrator then can check if access to such attribute is anomalous or normal and consequently modify the profiles.

D. Taxonomy of Query Anomalies

According to [13], query anomalies can be categorized into three types: 1) different schema/different tuples, 2) similar schema/different tuples, and 3) similar schema/similar tuples. In this subsection, we review the query types and show the types of anomalies that our AD techniques can identify.

A query is represented by its result set which has a schema, that is, the identifiers of the columns that appear in the result, and a set of tuples that verify the conditions expressed in the query. Two queries are considered different if they have different result schemas or if the set of tuples in the result of each query is statistically different from each other. In the rest of this subsection, we define and give examples on each type of anomaly. All examples follow the database schema shown in Table I.

1) Different Schema/Different Tuples: In this type of anomaly, anomalous queries have result sets whose result schema and tuples are different from those of normal queries. An example is when the normal query is (SELECT $*$ from students) and instead the query (SELECT $*$ from departments) is issued. This anomaly typically occurs in masquerading attacks. Our AD technique can detect this type of anomaly since the quadruplet contains information on the range tables and projection list.

2) Similar Schema/Different Tuples: Anomalous queries of this type have result sets whose schemas are similar to the result schemas of normal queries (that is, queries that are in the profiles) but have result tuples that are statistically different. By statistical difference we mean that if we computed statistical measures on columns, e.g., mean and standard deviation on integer columns, the resulting values of these measures would be different. This category has two subtypes. The first (subtype 2a) includes queries that have similar syntax. An example of this subtype is represented by the queries: (SELECT $*$ FROM students WHERE d_deptID $= 1$) and (SELECT $*$ FROM students WHERE d_deptID! $= 1$). The second subtype (2b) is when the normal and anomalous queries have different syntax. (SELECT $*$ FROM students where s_ID $= 1$) and (SELECT $*$ FROM students WHERE 1) are examples of such subtype. Data harvesting is an attack in which this type of anomaly occurs. Since the amount of data in the result of normal queries will be different than that in the anomalous ones, our system can detect anomalous queries of this type.

3) Similar Schema/Similar Tuples: This type is divided into two subtypes. The first (subtype 3a) includes queries that have different syntax and similar semantics. An example is represented by the queries (SELECT $*$ FROM students where s_ID $= 1$) and (SELECT $*$ from students where s_ID $= 1$ and s_deptID IN (SELECT d_ID from departments)). The two queries have similar semantics, i.e., user intent. Queries of this subtype should not be considered anomalous and indeed they would not be classified as such by our AD technique.

The second subtype (3b) is the case when the two queries have different syntax and different semantics too. An example is represented by the queries: (Select $*$ FROM students WHERE s_ID < 100) and (SELECT $*$ FROM students). In a rare case, all tuples stored in the table students will have $s_{\mathrm{ID}} < 100$ and all tuples will thus appear in the result. Therefore, the two queries will be equivalent. DBSAFE cannot distinguish between the two types of queries which is the correct decision.

V. Query Selectivity

An important issue in the design of our system is how to extract information for characterizing the data retrieved by queries. A critical requirement is that the query has to be analyzed before being submitted to the target RDBMS. Therefore, an approach based on analyzing the data returned by the query [13] would not be suitable. To address this issue, we rely on the query selectivity estimate provided by the query optimizer/planner inherent in most commercial RDBMS. In the rest of this section, we describe algorithms to estimate the selectivity, and requirements at the mediator level to support such algorithms.

A. Selectivity Estimation

Since an RDBMS usually has many execution plans to compute the answers to a query, the optimizer's job is to find

a good plan for evaluating the query by ordering the query operators and selecting the method for executing each operator. For example, consider a query that accesses a single table. The optimizer may choose to perform the projection operator before applying the where-clause condition or it may choose to apply the operators in the reverse order. As another example, the optimizer can choose to execute the join operator according to different algorithms, such as nested-loop join, merge-join, and hash-join. The output of the optimizer is a tree-structured plan for query execution; each node in the tree is an operator and the input(s) to this operator is the result of the child nodes. The choice of the optimizer on which plan to select is based on the cost of the query plan which is usually measured as the number of secondary storage page reads and writes. In the course of estimating the cost of a query plan, the optimizer also estimates the cost of each operator using statistics on tables and columns stored in the database catalogs. For DBSAFE, we adopt an approach that leverages the optimizer estimates on query selectivity as follows. We process the plan-tree of the query from top to bottom in a recursive manner in order to determine per-table selectivities using the selectivity estimates computed by the optimizer for the operators/internal nodes of the plan-tree. Our algorithm assumes plan-trees that are output of the PostgreSQL optimizer. As we process the plan-tree we compute the cardinality of each node which is the number of rows that are expected to be returned by the operation represented by this node. Examples of such operations include selection, projection, and join. The selectivity of a table that is accessed in the query is set to the minimum cardinality of the nodes in the path from the root of the tree to the leaf node that references this table, as computed by our algorithm, divided by the estimated table cardinality. Such cardinality is obtained from the table statistics maintained in the system catalogs.

We differentiate between two types of nodes in the plan-tree:

1) Nodes with a single input.

   A node that has one input represents a unary operator. Four unary operators exist in PostgreSQL: Sequential-Scan, Index-Scan, Materialize, and Hash operators. Input to Sequential-Scan and Index-Scan operators is a raw table, and they may have a filter condition. While parsing the plan-tree, when we encounter one such node, we set the cardinality of this node to the cardinality of the output of the node. The cardinality is recorded in the query-plan as part of the attributes of the node. The Materialize operator creates an in-memory table for some intermediate result to avoid reading a table or computing a result multiple times. A Hash operator is the left child of a Hash-Join operator and its function is to insert its input rows into a hash-table as part of the Hash-Join algorithm. The selectivity of both the Materialize and the Hash operators is obtained from the node attributes and used further to compute the minimum values across paths from the root to the leaves that pass through this node as described earlier.
2) Nodes with two inputs.

   These are join operators for which we differentiate between two types as follows.

   a) Equi-Join on a specific attribute.

   An example of such node is the root of the plan-tree of the query "Select $*$ from R_1, R_2 where $R_1.A = R_2.A$". We distinguish here between two cases:

   i) $R_2.A$ is a foreign-key to the primary-key $R_1.A$.

      In this case, we assume that all rows of R_2 will show in the result. The number of null values in $R_2.A$ are excluded if the join type is not a left outer join. The cardinality of the node relative to the subtree that references R_1 is set to the number of distinct values in $R_2.A$, if this information is present in the data dictionary and 1 otherwise. If the join has a filter condition, the cardinality of the node seen by the left and right subtree is set to the output cardinality of the node.

   ii) There is no such relationship between the columns. In this case we differentiate between two cases:

   A) The join is an inner join, and both $R_1.A$ and $R_2.A$ have most common values (MCVs) statistics recorded in the data dictionary.

      For $i \in \{1, 2\}$, let r_i be the number of rows of R_i, nd_i be the number of distinct values in $R_i.A$, n_mcv_i be the number of most common values in $R_i.A$, $n\_d\_\text{mcv}_i$ be the number of distinct values in the MCVs of R_i, n_null_i be the number of nulls in $R_i.A$, and $\text{null\_}f_i$ be the fraction of rows that have nulls in $R_i.A$. All these values can be read directly from the statistics of the columns stored in the data dictionary of PostgreSQL.

      To compute the node cardinality, first, we find the common values in both arrays of MCVs of the columns; let these values be vm and their number be nm_i. The rest of the MCVs of R_1 can match any value in R_2 other than those in mcv_2; the number of distinct values in this portion of the table is $n\_d\_\text{mcv}_1 - |vm|$. The number of rows that would match each of these values in R_2 will be $(r_2 - \text{null\_}f_2 - n\_\text{mcv}_2)/(nd\_2 - n\_d\_\text{mcv}_2)$. The rest of the R_1 rows can match any value in the non-common values in R_2 and the common values in R_2 other than vm; the number of distinct values in this portion of R_1 is $nd_1 - n\_d\_\text{mcv}_1$ and the number of rows that match each of these values in R_2 is $(r_2 - \text{null\_}f_2 - nm_2)/(nd_2 - |vm|)$. Using the previous formulas, the cardinality of R_2 in the result can be computed if we assume that all the rows of R_1 will appear in the result. The same idea applies if all rows in R_2 are assumed to appear in the result. We calculate the cardinalities of both tables given the previous two assumptions; the assumption that yields a higher cardinality value of the table will be assumed true. The cardinality of the join node relative to each subtree is computed accordingly.

   B) The above case does not apply.

      In this case, the selectivity of R_1 and R_2 in Q is set to $(1 - \text{null\_}f_1)$ and $(1 - \text{null\_}f_2)$ respectively. The null fraction is only excluded from R_1, R_2 or both in the case the type of join is not left, right or full, respectively.

TABLE III
DESCRIPTION OF TRAINING DATA AND EVALUATION RUN

Training Database
- 30 Tables
- Avg. 18 columns per table
- Avg. 600,000 rows per table
- 3 roles (doctors, nurses, hospital administrators)
- 7 unique DB userIDs
Training Queries
- 10,000 - 12,000 queries per role
Evaluation Run
- Approx. 6 hours
- 7 users (4 human, 3 automated via JMeter)
- Approx. 520 total queries
- Approx. 40 exfiltration attempts

b) Other types of join.

In this case, all rows of both inputs are assumed to appear in the result of this node and the cardinality of the node used to compute the minimum of the left table is the estimated number of rows in the output of the left subtree of the join node. If the left node is a raw table, this value is obtained from the data dictionary. Similarly, the cardinality of the node as seen by the right subtree is the estimated number of rows of the right subtree of the node.

B. T-DBMSs Adapters

It is important to note that the ADE optimizer requires statistics on the actual data in order to generate a suitable plan-tree for the query. These statistics are generally only available on the target RDBMS since this is the system managing the actual data. PostgreSQL stores the most common values and height-balanced histograms for each table column. For RDBMSs, like Oracle, that adopt the same type of optimizer statistics, we can directly use the statistics already existing in the Oracle data dictionary rather than computing these values from scratch. For SQL Server and MySQL, which do not store height-balanced histograms, we need to compute this data using aggregate functions. Therefore, DBSAFE needs to integrate an adapter for each type of RDBMS to be supported. Currently, we have developed adapters for Oracle and SQL Server.

VI. EXPERIMENTAL ACTIVITIES

DBSAFE was evaluated over several weeks in the summer of 2014. The test database used the schema corresponding to a real-world government medical database used to train doctors and other personnel but populated with artificially generated data. Table III summarizes the database used for evaluation, the training log, and the evaluation run. In order to test the efficacy of DBSAFE, our team emulated a government production system, supplementing human security operators with automated processes. One team, designated the Blue Team, was responsible for monitoring the system and protecting sensitive data using standard security techniques leveraging the DBSAFE prototype software, while another team, designated the Red Team, performed normal actions associated with the production system. Select members of the Red Team, whose identities were unknown to the Blue Team, attempted to exfiltrate sensitive data periodically during the evaluation test run and/or to cover up nefarious activities. A third team, designated the White Team, monitored the evaluation process and gathered metrics for performance analysis. Fig. 2 shows the test environment.

The Red Team was supplemented by two background workloads injected via the Apache JMeter load-testing tool. One workload consisted entirely of benign database queries; those are queries consistent with the tasks/activities of roles and thus were used previously for training. The second workload consisted entirely of queries previously developed by the evaluation team to exfiltrate data. These exfiltration queries fell into two categories. The first category consists of SELECT statements which retrieve very sensitive columns of personally identifiable information, such as social security number or passport number. The second category consists of INSERT statements that create new actors in the system. Creating new users is a typical activity for malicious insiders and outsiders masquerading as insiders.

To provide a baseline, the system was run without DBSAFE to determine how well Blue Team members could recognize exfiltration attempts without the automated support of DBSAFE. The results were surprising: not a single true data exfiltration attempt was recognized by any member of the Blue Team, despite using a small set of SQL statements and lengthy time between successive SQL statements.

Then we performed the actual evaluation run where duplicate SQL statement streams were routed to both classifiers simultaneously. Table IV reports the results of this evaluation and shows the accuracy of the Bayesian classifier and the Binary classifier. The column named Bayesian classifier with warnings shows the accuracy of the Bayesian classifier when queries that produc ed a warning were considered anomalous. Remember that a warning will be produced for each attribute referenced in the query but has never appeared in the training log of the role of the user, see Section IV-C. Table V contains the true positive rate (TPR) and true negative rate (TNR) for both types of classifiers.

As can be seen, both classifiers performed well with respect to true positives, at the cost of unacceptably high false positives. However, the NBC consistently outperformed the Binary classifier. The reader may note that, though the same set of SQL statements were sent to both classifiers, the two classifiers reported different numbers of statements processed.

As shown in Table V, the true negative rates of the NBC (with warnings) and the binary classifier are 68.36% and 29.53%, respectively. Both classifiers have high rate of false positives (Note that the false positive rate (FPR) is equal to 1—TNR); the reason is that the training logs did not have enough data about the columns that are usually retrieved by queries that are considered normal. The result is that the system would flag queries, that are considered benign by the Blue team member, as anomalous. Therefore, it is important to have training data that covers all the access patterns that are considered normal. To validate that the reason for the false positives is the incompleteness of the training data set and to better understand other relevant factors, we ran several experiments using synthetic data. Our experiments show that the cause for the false positives of the NBC is a combination of insufficient training data, different numbers of training records available for the different roles, and common access patterns between the roles. The results show that, for a database of 10 tables, with five columns per table, and five roles, when the length of the training log of one

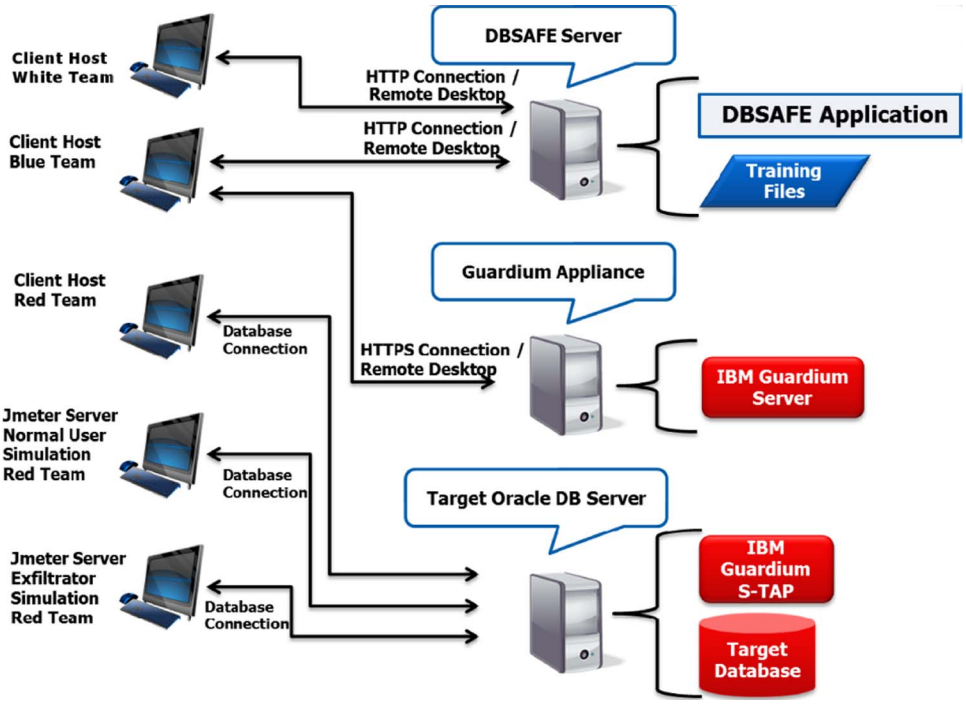

Fig. 2. DBSAFE evaluation environment.

TABLE IV
EVALUATION RESULTS

	Ground Truth	Bayesian Classifier		Bayesian Classifier with Warnings		Ground Truth	Binary Classifier	
		Raw Number	Percentage	Raw Number	Percentage		Raw Number	Percentage
True Positives	44	29	65.91%	43	97.73%	39	31	79.49%
True Negatives	414	339	81.88%	283	68.36%	359	106	29.53%
False Positives	0	75	18.12%	131	31.64%	0	253	70.47%
False Negatives	0	15	34.09%	1	2.27%	0	8	20.51%

TABLE V
TRUE POSITIVE AND TRUE NEGATIVE RATES

	Bayesian	Bayesian with warnings	Binary
TPR	65.91%	97.73%	79.49%
TNR	81.88%	68.36%	29.53%

role is less than 10% of the length of the log of the role that has the maximum number of training records, accompanied by having more than 10% common queries between the roles, the detection result is not accurate and queries from roles with short logs will always be considered anomalous.

One solution to the problem of insufficient training data is to support active learning techniques in which feedback from the security administrator is incorporated in the system so that it continuously learns about normal queries. In the future work section, we discuss solutions to the problems of common role access patterns and the variations in the length of the training logs of the different roles.

Another observation on the results is that the Binary classifier generates a higher number of false positives compared with the NBC. The reason is that the former is not able to deal well with the case in which the issued queries have some minor variations with respect to the queries used for training. For example, suppose that the training data contains, for one of the roles in the database described in Section III, the query "Select s_ID from students." If a user who belongs to this role issues the query "Select s_ID, s_deptID from students," the Binary classifier will classify this query as anomalous; on the other hand, the NBC will classify it as a normal query as long as this role has a higher probability of sending such query compared with the other roles. Notice that, independently from the specific classifier used, DBSAFE would also generate a warning indicating that the query references an attribute that has never been accessed by the issuing role in the training log. The security administrator can then inspect the attribute identified in the warning and let DBSAFE know whether the access to this attribute by the role issuing the query is normal or has to be considered an anomalous access. Therefore, subsequent accesses to this attribute will be automatically classified by DBSAFE according to such indication.

VII. RELATED WORK

Several approaches have been proposed to detect anomalies in database system access. Spalka *et al.* [16] propose a misuse detection system for databases. They compare two approaches for performing AD. The first is to compute reference values on monitored attributes. Periodically, values of attributes are checked against reference values, and an anomaly is raised if the difference exceeds a threshold set by the user. One problem with this method is that it cannot be used to prevent an attack before it occurs. Another method was proposed to address this problem, namely, to store Δ-relations that capture the changes performed on the monitored tables and periodically

apply these changes on fictitious relations that are similar to the monitored ones and then apply the first approach on these fictitious relations to detect anomalies. Both approaches require large processing overhead; they also focus on updates and therefore are unable to detect anomalies in read accesses. Failure to capture the behavior of different users is also a shortcoming in both approaches. Chung et al. propose DEMIDS, a misuse detection system for relational database systems [4]. DEMIDS uses data in audit logs to derive profiles describing typical patterns of access by database users. They introduced the concept of frequent itemsets representing queries that are frequently issued to the database. The proposed approach has the same problem of not differentiating between the behaviors of different users. The authors did not provide experimental results to show the effectiveness of the proposed approach.

Other systems have been designed that work at the database application level. For example, DIDAFIT [11] is a system for detecting illegitimate database accesses by matching statements against a known set of legitimate database transaction fingerprints. DIDAFIT approach is based on summarizing SQL queries into compact regular expressions. One problem in such method is that the order of commands in the program is not taken into consideration. DetAnom [7] overcomes the problem in DIDAFIT by using concolic testing techniques to capture the control flow and data flow of the program to assure the correct order of SQL commands is preserved in the user transactions. The use of data access correlations (read and write sets) has been proposed in order to detect anomalies in one or multiple consequent transactions [3], [6]. Since these approaches are designed to work at the application level, they cannot differentiate between the individual users/roles.

Approaches complementary to ours have been proposed [12], [18], and [19] to determine the amount of knowledge that can be inferred about database objects using dependency relationships [12], [18], and [19]. This knowledge can be used by insiders to broaden their knowledge about the database. Other approaches [17] and [20] focus on analyzing network traffic to detect malware that could lead to data exfiltration. All these approaches can be combined with ours in order to strengthen data protection from insiders.

Our current work is based on earlier ideas by Bertino *et al.* [2], Kamra *et al.* [10], and Shebaro *et al.* [15]. This previous work introduced the idea of profiling query syntax to detect anomalies in queries but has several limitations, including failing to characterize the amount of data returned by queries, requiring modifications to the source code of the target DBMS, and lack of integration with SIEM tools.

VIII. On-Going and Future Work

We are currently working on extending the DBSAFE system in different directions. The first is adding support for profiling application programs, as in typical n-tier architectures, intermediary application programs access the DBMS as opposed to direct access by human users. A malicious insider may change the code of the application program or submit malformed input to cause a SQL injection attack in order to exfiltrate data. However, creating query profiles for application programs is challenging since specific transactions can only be issued by the program according to its control and data flows. Moreover, strings of queries in the same control flow sequence of the program may be different if constants in the query strings are computed based on input values. To address these issues, we have developed an approach based on a combination of concrete testing and symbolic execution techniques, referred to as concolic testing [7], [14]. This technique works in both the profile creation and the detection phases. During the profile-creation phase, we use concolic testing to build a profile for the program in which the control structure of the program and locations in the code where SQL queries are issued are recorded. The profile also records expressions needed to compute constants in the program in terms of input values. During program runtime (the detection phase), the system uses this profile and the input values to compose exact query strings that are expected to be issued by the program. These strings are compared to the actual ones sent by the program to the RDBMS; any difference is considered anomalous. An interesting research direction, that we plan to pursue, is to apply such approach to desktop applications, web applications and web services. We will also investigate how to use DBSAFE to detect spyware and other malware that try to access data by exploiting data privileges of users.

A second critical extension is related to supporting AD when roles are either not available or not used, even if they are available. The latter case can arise since currently available RDBMSs allow users to initially login using their user ID and then choose a different role. Therefore, using a role is not necessarily mandatory. Thus, it is also important to profile individual user behavior. To address this issue, we are investigating two complementary approaches. The first approach is to create a role dedicated to each single user, in addition to the normal DBMS roles. This special role would enable profiling the access patterns of its corresponding user and comparing current access patterns by the user against this profile. The second approach is to cluster users that are characterized by the same access patterns and create a role for this cluster. The two approaches could complement each other in that the single-user role approach would allow one to compare the activities of a user against his/her past profiled activities. Upon detecting anomalous queries by a user, one could check if such anomalies occur for other users in the same cluster. If so, the change in access patterns could be recorded as a normal event due to changes in situations/tasks. Along similar lines, another important function to include is to model and use different profiles for different organizational situations, for example normal versus emergency situations.

A third critical extension is related to training data. We identified two major issues in the use of classification for role prediction. The first issue is due to imbalanced training data when there are significant differences between the cardinalities of training records for different roles. Training models using such imbalanced data will lead to sub-optimal predictive solutions. To address this issue, we are extending DBSAFE with the following approach. When the training phase ends, the mediator checks the number of queries submitted by each role. If there is a difference of more than α between the number of queries of a role and the maximum, the mediator sends additional queries

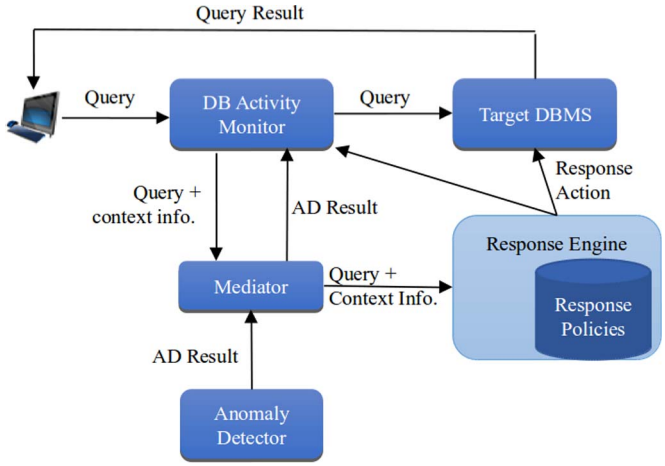

Fig. 3. Supporting response actions.

to the ADE by sampling (with replacement) the actual queries submitted by the users of this role. A second issue related to training data when using the NBC with the MAP rule is that if users submit similar queries within different roles, the MAP rule will not produce proper results if these same queries are submitted during detection. The reason is that the probability that any of these queries occurs will be biased toward the role that submitted this query or part of it more frequently than the other roles. Two techniques can be used to solve this problem. The first is to identify the features that best represent each class/role of the classifier and select only these features to be used in classification. This technique is the well-known feature selection used for bootstrapping classifiers. Several feature selection algorithms have been proposed. We plan to use the minimum redundancy maximum relevance (mRMR) feature selection algorithm [5]. This algorithm tries to find features that are the most relevant or best represent the available classes, and at the same time to omit features that give the same information (entropy) as others, thereby reducing the redundancy among selected features. The second technique that can be used to solve the problem of common access patterns between roles is to use a family of classifiers called the multi-labeling classifiers. Using multi-labeling classification, in contrast to the NBC with the MAP rule which relates each query to exactly one role, a query can be related to several roles if it is a commonly sent by the users of different roles.

A fourth critical extension is to support response actions for handling detected anomalies. How to handle a specific anomaly may depend on many factors. For example, if a table being accessed contains very sensitive data, a strong response to the anomaly would be to revoke the privileges corresponding to anomalous accesses. In other cases, it may be desirable to take no response actions in order not to tip the insider that he/she has been detected. Our previous work [9] considers using a response engine that automatically determines the action to execute based on information about the anomaly and a response policy base. This approach assumes that the target RDBMS is PostgreSQL type and has an AD component. We are currently extending our previous work to support the new architecture that separates the AD task from the target RDBMS. Fig. 3 shows the flow of information to support response actions. The response engine returns actions for responding to the anomaly based on the query, the AD result, and contextual information associated with the query. These actions are selected from a set of predefined possible responses expressed in a declarative language that we refer to as response policies.

Finally, future work includes designing AD techniques for non-SQL DBMS and to use fine-grained provenance techniques to monitor the use of data by users and applications once the data have been retrieved.

ACKNOWLEDGMENT

The work reported in this paper has been funded in part under contract by Department of Homeland Security (DHS) Science and Technology Directorate, Homeland Security Advanced Research Projects Agency, Cyber Security Division. We would like to thank Lorenzo Bossi of Purdue University for his work on re-engineering the DBSAFE Proof of Concept, Dorota Woodbury, Dave Lafave, Catherine LaShure, and Mark Pumphrey at Northrop Grumman for their work on extending and integrating the DBSAFE prototype with commercial systems, and our numerous colleagues at Northrop Grumman for setting up and participating in the evaluation and testing exercises.

REFERENCES

[1] E. Bertino, "Data protection from insider threats," *Synthesis Lectures Data Manage.*, vol. 4, no. 4, pp. 1–91, 2012.
[2] E. Bertino, A. Kamra, E. Terzi, and A. Vakali, "Intrusion detection in rbac-administered databases," in *Proc. 21st ACSAC*, Washington, DC, USA, 2005, pp. 170–182.
[3] M. Chagarlamudi, B. Panda, and Y. Hu, "Insider threat in database systems: Preventing malicious users' activities in databases," in *Proc. 6th Int. Conf. ITNG*, Apr. 2009, pp. 1616–1620.
[4] C. Y. Chung, M. Gertz, and K. Levitt, "Integrity and internal control information systems," in *DEMIDS: A Misuse Detection System for Database Systems*. Norwell, MA, USA: Kluwer, 2000, pp. 159–178.
[5] G. Gulgezen, Z. Cataltepe, and L. Yu, "Stable and accurate feature selection," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, vol. 5781, W. Buntine, M. Grobelnik, D. Mladeni, and J. Shawe-Taylor, Ed. Berlin, Germany: Springer-Verlag, 2009, pp. 455–468.
[6] Y. Hu and B. Panda, "Identification of malicious transactions in database systems," in *Proc. 7th Int. Database Eng. Appl. Symp.*, Jul. 2003, pp. 329–335.
[7] S. R. Hussain, A. M. Sallam, and E. Bertino, "Detanom: Detecting anomalous database transactions by insiders," in *Proc. 5th ACM CODASPY*, New York, NY, USA, 2015, pp. 25–35.
[8] IBM, IBM—InfoSphere Guardium Data Activity Monitor. Accessed: May 1, 2015. [Online]. Available: http://www-03.ibm.com/software/products/en/infosphere-guardium-data-activity-monitor
[9] A. Kamra and E. Bertino, "Design and implementation of an intrusion response system for relational databases," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 6, pp. 875–888, Jun. 2011.
[10] A. Kamra, E. Terzi, and E. Bertino, "Detecting anomalous access patterns in relational databases," *Int. J. Very Large Data Bases*, vol. 17, no. 5, pp. 1063–1077, Aug. 2008.
[11] S. Lee, W. Low, and P. Wong, "Learning fingerprints for a database intrusion detection system," in *Computer Security ESORICS 2002*, ser. Lecture Notes in Computer Science, vol. 2502, D. Gollmann, G. Karjoth, and M. Waidner, Eds. Berlin, Germany: Springer-Verlag, 2002, pp. 264–279.
[12] W. Li, B. Panda, and Q. Yaseen, "Mitigating insider threat on database integrity" in *Information Systems Security*, ser. Lecture Notes in Computer Science, vol. 7671, V. Venkatakrishnan and D. Goswami, Eds. Berlin, Germany: Springer-Verlag, 2012, pp. 223–237.

[13] S. Mathew, M. Petropoulos, H. Q. Ngo, and S. Upadhyaya, "A data-centric approach to insider attack detection in database systems," in *Proc. 13th Int. Conf. RAID*, Berlin, Germany, 2010, pp. 382–401.
[14] A. Sallam and E. Bertino, "Poster: Protecting against data exfiltration insider attacks through application programs," in *Proc. ACM SIGSAC Conf. CCS*, New York, NY, USA, 2014, pp. 1493–1495.
[15] B. Shebaro, A. Sallam, A. Kamra, and E. Bertino, "Postgresql anomalous query detector," in *Proc. 16th Int. Conf. EDBT*, New York, NY, USA, 2013, pp. 741–744.
[16] A. Spalka and J. Lehnhardt, "A comprehensive approach to anomaly detection in relational databases," in *Data and Applications Security XIX*, ser. Lecture Notes in Computer Science, vol. 3654, S. Jajodia and D. Wijesekera, Eds. Berlin, Germany: Springer-Verlag, 2005, pp. 207–221.
[17] H. Xiong, P. Malhotra, D. Stefan, C. Wu, and D. Yao, "User-assisted host-based detection of outbound malware traffic," in *Proc. 11th ICICS*, Berlin, Germany, 2009, pp. 293–307.
[18] Q. Yaseen and B. Panda, "Knowledge acquisition and insider threat prediction in relational database systems," in *Proc. Int. Conf. CSE*, Aug. 2009, vol. 3, pp. 450–455.
[19] Q. Yaseen and B. Panda, "Predicting and preventing insider threat in relational database systems," in *Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices*, ser. Lecture Notes in Computer Science, vol. 6033, P. Samarati, M. Tunstall, J. Posegga, K. Markantonakis, and D. Sauveron, Eds. Berlin, Germany: Springer-Verlag, 2010, pp. 368–383.
[20] H. Zhang, D. D. Yao, and N. Ramakrishnan, "Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery," in *Proc. 9th ACM Symp. Inf., ASIA CCS*, New York, NY, USA, 2014, pp. 39–50.

Asmaa Sallam is pursuing the Ph.D. degree at the department of computer science at Purdue University, West Lafayette, IN, USA.

Her research interests are on data protection from insider threat and data management systems.

Elisa Bertino (SM'83–F'02) received the Ph.D. degree in computer science from the University of Pisa, Pisa, Italy.

She is a Professor of computer science at Purdue University, and serves as Research Director of the Center for Education and Research in Information Assurance and Security (CERIAS) and Interim Director of Cyber Center (Discovery Park). Previously, she was a Faculty Member and Department Head at the Department of Computer Science and Communication of the University of Milan. She is currently serving as EiC of IEEE Transactions on Dependable and Secure Computing. She is a Fellow of the ACM.

She received the 2002 IEEE Computer Society Technical Achievement Award for outstanding contributions to database systems and database security and advanced data management systems and the 2005 IEEE Computer Society Tsutomu Kanai Award for pioneering and innovative research contributions to secure distributed systems.

Syed Rafiul Hussain is pursuing the Ph.D. degree at the department of computer science at Purdue University, West Lafayette, IN, USA.

His research interests are on data protection from insider threat and provenance techniques for sensor networks.

David Landers received the B.S. degree in physics from Florida Institute of Technology, Melbourne, in 1995.

He is a Software Engineer in the Information Systems Sector of Northrop Grumman Corporation in Northrop Grumman Corporation. From 1995 to 2008, he was a Database Administrator and Software Engineer with L-3 Communications Inc. Since 2008 he has been a Database Administrator, Systems Architect, Test Engineer, and Software Engineer at Northrop Grumman. His current fields of interest are data warehouse design and database security.

R. Michael Lefler received the B.S. degree in mathematics and M.S. degree in computer science from University of Illinois, Urbana, IL, USA.

He is a Computer Systems Architect and Technical Fellow of the Enterprise Shared Services organization of Northrop Grumman Corporation, where he is responsible for predictive analysis applied to the Insider Threat. At Northrop Grumman he has architected advanced information system solutions in the intelligence community, DoD, law enforcement, and healthcare and served as Principal Investigator on numerous R&D projects in areas such as advanced healthcare, database performance prediction, and information assurance. From mid-1996 through December 2001 Mr. Lefler was a voting member of the American National Standards Institute (ANSI) INCITS H2 Committee.

Donald Steiner received the Ph.D. degree in mathematics from Iowa State University, Ames.

He is the Principal Technologist and a Technical Fellow of the Information Systems Sector of Northrop Grumman Corporation. He is Principal Investigator for research and development projects involving data analytics, cybersecurity, and cloud computing. He manages the Northrop Grumman Cybersecurity Research Consortium, a collaboration with Carnegie Mellon University, MIT, Purdue University, and the University of Southern California. Previously, he was CTO of Quantum Leap Innovations and Co-Founder and Chief Scientist of WebV2, Inc., a spin-off of Siemens AG. His research interests include artificial intelligence, multi-agent systems, data analytics, cloud computing, and cyber security.