A Data-Centric Approach to Insider Attack Detection in Database Systems

Sunu Mathew [#1], Michalis Petropoulos [#2], Hung Ngo [#3], Shambhu Upadhyaya [#4]

[#]*Department of Computer Science and Engineering, University at Buffalo*
Buffalo, NY 14260, USA
[1]smathew2@buffalo.edu
[2]mpetropo@buffalo.edu
[3]hungngo@buffalo.edu
[4]shambhu@buffalo.edu

***Abstract*— The insider threat against database management systems is a very dangerous and common security problem. Authorized users may compromise database security by abusing legitimate privileges to masquerade as another user or to gather data for malicious purposes.**

This paper proposes a direction to solve this problem: profiling user database access patterns by looking at exactly what the user accesses. The approach is data-centric in the sense that query expression syntax is considered irrelevant for discriminating user intent – only the resulting data matters in this regard. Several questions arise – what do we mean by "profiling"? How do we implement this idea and how effective is the solution?

We answer these two questions by outlining a method to model a users' database access with a multidimensional vector. Statistical learning algorithms are then trained and tested on these vectors, using real data from a Graduate Admission database. Several performance issues are also addressed. Experimental results indicate that the technique is very effective, accurate, and is promising in complementing existing database security solutions.

I. Introduction

Ensuring the security and privacy of data assets is a crucial and very difficult problem in our modern networked world. Relational database management systems (RDBMS) [1] is the fundamental means of data organization, storage and access in most organizations, services, and applications. Naturally, the prevalence of RDBMS also led to the prevalence of security threats against database management systems.

An intruder from the outside, for example, may be able to gain unauthorized access to data by trying to execute well-crafted queries against a back-end database of a Web application. This class of so-called *SQL injection* attacks [2] are well-known and well-documented, yet still very dangerous [3]. They can be mitigated by adopting suitable safeguards, for example, by adopting defensive programming techniques and by using SQL *prepare* statements [4].

An *insider attack* against a database management system, however, is much more difficult to detect, and potentially much more dangerous [5]–[7]. According to the most recent Secret Service/CERT/Microsoft E-Crime report, insider attacks constitute 34% of all surveyed attacks (outsiders constitute 37%, and the remaining 29% of surveyed attacks have unknown sources). For example, insiders to an organization such as (former) employees or system administrators might abuse their *already existing privileges* to conduct *masquerading*, *data harvesting*, or simply *sabotage* attacks [8].

Somewhat more formally, an insider (inside attacker) is typically viewed as someone with legitimate privileges operating with malicious intent. The RAND workshop devoted to insider threats [9] defined an insider as "someone with access, privilege or knowledge of information systems and services," and the insider threat problem as "malevolent (or possibly inadvertent) actions by an already trusted person with access to sensitive information and information systems." For example, attacks such as *masquerading* and *privilege abuse* represent well-known security threats in financial, corporate and military domains; attackers may *abuse legitimate privileges* to conduct *snooping*, *data-harvesting* [5] and other actions with malicious intent (e.g., espionage). Detecting insider attacks by specifying explicit rules or policies is a moot point – an insider is always defined *relative* to a set of policies.

Consequently, we believe that the most effective method to deal with the insider threat problem is to statistically profile normal users' (computing) behaviors and raise a flag when a user deviates from his/her routines. Intuitively, for example, a good statistical profiler should be able to detect non-stealthy sabotage attacks or quick data harvesting attacks because the computing footprints of those actions should be significantly statistically different from the day-to-day activities.

The user profiling idea for insider threat detection in particular and anomalous detection in general is certainly not new (see, e.g., [10]). In the context of an RDBMS (or any problem requiring statistical profiling), the novelty lay in the answers to two critical questions:

(1) what is a user profile? and
(2) which statistical/machine-learning techniques and models to be adopted so that the profiles are *practically useful* for the detection problem?

By "useful" we mean some large class of insider attacks can be detected. By "practical" we mean the method can be deployed and perform effectively in a real RDBMS. The novelty and contributions of this paper come from answering the above two questions.

Several research projects (e.g., [11]–[16]) have led to the development of intrusion detection (ID) and mitigation systems that specifically aim to protect databases from attacks. Our work in this paper will focus on analyzing user behavior specifically in terms of SQL queries to a relational database. Other behavioral features that should be useful in insider threat detection (e.g., location of the attacker, information correlation between consecutive queries, and temporal features such as time between queries, duration of session etc.) are outside the scope of this paper, and will certainly be considered in a future work.

Perhaps the most natural user "profile" is the set of SQL queries a user daily issue to the database, or some representative set of queries representing the past. For example, [16] relied on the SQL-expression syntax of queries to construct user profiles. This approach has the advantage that the query processing of the intrusion/insider detection system is computationally light: a new query is analyzed and run through some statistical model (e.g., clustering) and only queries that are accepted by the detection system are then actually executed in the database engine. However, as we shall later demonstrate in this paper, this syntax-centric view is ineffective and error-prone for database anomaly detection in general, and for database insider threat detection, in particular. For example, two queries may differ widely in syntax and yet produce the same "normal" (i.e., good) output. Thus, a syntax-based detection engine might generate false positives on these queries. Conversely, two queries may be very similar in syntax and yet generate completely different results, causing the syntax-based engine to generate false negatives.

Our conviction is that the best way to distinguish normal vs. abnormal (or good vs. malicious) access patterns is to look directly at *what* the user is trying to access – the result of the query itself – instead of *how* he expresses it, i.e. the SQL expressions. Two syntactically different queries with similar result tuples should be considered the same (whether good or malicious) by the threat detection engine; conversely, two syntactically similar queries should be considered different if they result in different tuple sets. An insider who tries to peek at a part of the database which he often does not access will be caught. The same holds for an attacker using a compromised account as the attacker's data access pattern will likely be different from the real account owner's access pattern. Data browsing for harvesting purposes would also be caught if browsing is not part of his daily routines.

The conviction is the core of our answer to question (1) above; our approach is data-centric rather than syntax-centric: user behavior is modeled on the basis of the data generated by executing their queries as opposed to the syntax of the SQL expressions. We shall show that the data-centric approach is superior in terms of detecting anomalous user access patterns. Technically, however, it is still not quite clear what a user profile is. A typical database may consist of millions of data tuples, it is certainly impractical and in fact may not even be useful to explicitly keep track of all data tuple sets that are the results of "normal" queries. Briefly, we solve this problem by viewing the data as a universal-relation [17]. Each query result (a set of tuples) will be represented by a compact vector whose dimension is fixed, regardless of how large the query's result set is. More details will be given in Section IV.

Intuitively, this approach has pros and cons. On the plus side, for an insider to evade our system, he would likely have to generate queries with the same (or statistically similar) result set as the result sets he would have gotten anyhow with his legitimate queries using his existing privileges, rendering the attempt at circumvention inconsequential. In the syntax-based approach, queries with similar syntax can give different results and the attacker may be able to craft a "good-looking" malicious query to access data he's not supposed to access.

On the minus side, a query has to be executed *before* the decision can be made on whether or not it is malicious. What if a malicious query asks for hundreds of gigabytes of data? Will the query have to be executed, and will our detection engine have to process this humongous "result set" before detecting the anomaly? These legitimate concerns are within the scope of question (2) above. We will show that this performance-accuracy tradeoff is not at all as bad as it seems at first glance. We will experimentally show that a representative constant number of result tuples per query are sufficient for the detection engine to perform well, especially when the right statistical features and distance function (between normal and abnormal result sets) are chosen. Furthermore, these (constant number of) result tuples can be computed efficiently by leveraging the pipelined query execution model of commercial RDBMSs. Feature selection and the distance function choice are among the key contributions of this paper.

The rest of this paper is organized as follows. Section II surveys background and related works. Section III demonstrate technically the limitations of the syntax-based approach, motivating the data-centric approach introduced in Section IV. Section V gives a brief taxonomy of query anomalies facilitating the experiments presented in Section VI. We further discuss our solution, its implications, and future research directions in Section VII.

II. Background and Related Work

Several intrusion detection systems (IDS) with direct or indirect focus on databases have been presented in the literature. Generic approaches and architectural frameworks for database IDS have been proposed in [18] and [19]. In [20], the real-time properties of data are utilized for intrusion detection in applications such as real-time stock trading. Kreugel et. al. [21] and Valeur et. al. [14] present schemes for the detection of anomalies such as SQL-injection attacks in web-based applications. We believe that detection of sql-injection attack is a specific kind of database query anomaly that is detected by our approach in a straightforward manner as we will explain in this paper.

Data dependency among transactions is used to aid anomaly detection in [13] – the central idea here is that database transactions have data access correlations that can be used for intrusion detection at the transaction and user task levels.

Similarly, in [22], the concept of dependency between database attributes is used to generate rules based on which malicious transactions are identified. The DEMIDS system [11] detects intrusions by building profiles of users based on their working scopes which consist of feature/value pairs representing their activity. These features are typically based on syntactical analysis of the queries. A system to detect database attacks by comparison with a set of known legitimate database transactions is the focus of [12]. SQL statements are summarized as regular expressions which are considered to be 'fingerprints' for legitimate transactions – this again, is based on analysis at the syntactical level. In [23], an approach to intrusion detection in web databases is proposed that is based on constructing fingerprints of all sql statements that an application can generate. A binary vector with length equal to the number of fingerprints is used to build session profiles and aid in anomaly detection. This approach introduces assumptions such as restrictions on the number of distinct queries possible, and may complement our approach in cases where the assumptions are valid. In [24], database transactions are represented by directed graphs describing the execution paths (select, insert, delete etc.) and used for malicious data access detection. This approach cannot handle adhoc queries (as the authors themselves state) and works at the coarse-grained transaction level as opposed to the fine-grained query level. Database session identification is the focus of [25] – queries within a session are considered to be related to each other, and an information theoretic metric (entropy) is used to separate sessions; however, whole queries are used as the basic unit for n-gram-statistical modeling of sessions. A multiagent based approach to database intrusion detection is presented in [26]; relatively simple metrics such as access frequency, object requests and utilization and execution denials/violations are used to audit user behavior.

Prior approaches in the literature that have the most similarity to ours are [15] and [16]. The solution for database anomaly detection proposed in [15] is similar in the use of statistical measurements; however the focus of the approach is mainly on detecting anomalies in database modification (e.g., *inserts*) rather than queries. The query anomaly detection component is mentioned only in passing and only a limited set of features (e.g., session duration, number of tuples affected) are considered. The work presented recently in [16] has the same overall detection goals as our work here – detection of anomalies in database access by means of user queries. However, it takes an approach that is based on analyzing the syntax of sql strings for anomaly detection, unlike our approach of analyzing the results of query execution. A primary focus on this paper will be on exposing the limitations of syntax based detection schemes; the approach in [16] will be used in this paper as a benchmark for evaluating the performance of our approach.

III. Limitations of Syntax-Centric Approach

This section presents examples to demonstrate the limitations of the syntax-centric approach, showing that two syntactically similar queries may generate vastly different results, and two syntactically distinct queries may give similar results. Consequently, statistical profiles based on SQL expressions are limited in their ability to recognize users' intents. For example, a syntax-based approach may model a query with a frequency or characteristic vector, each of whose coordinates counts the number of occurrences (or marks the presence) of some keywords (e.g., `select, from, etc.`) or mathematical operators [16].

Consider the following query:

```
SELECT p.product_name, p.product_id
FROM PRODUCT p
WHERE p.cost == 100;
```

A syntactical analysis of this query and subsequent feature extraction (e.g., [16]) might result in the following features for query data representation – SQL Command – *SELECT*, Select Clause Relations – *PRODUCT*, Select Clause Attributes – *product_name, product_id*, Where Clause Relation – *PRODUCT*, Where Clause Attributes – *cost*. Now consider the alternate query:

```
SELECT p.product_name, p.product_id
FROM PRODUCT p
WHERE p.cost != 100;
```

This query has the same data representation (based on syntax-analysis) as the previous one; however, it is easy to see that the data tuples accessed in the two cases are very different (in fact, they are the complement of each other).

Similarly, consider the first query rewritten as follows:

```
SELECT p.product_name, p.product_id
FROM PRODUCT p
WHERE p.cost == 100
AND  p.product_name is not null;
```

This query has a different syntax (two columns and a conjunction operator in the WHERE clause), but produces the same result tuples as the first (under the reasonable assumption that all products in the database have a valid product name). Most syntax-based anomaly detection schemes are likely to flag this query as an anomaly with respect to the first.

Syntax analysis, even if very detailed (taking into account differences in operators, e.g., '==' and '!=' in the examples above) is complicated given the richness of the SQL language, and involves determining *query equivalence*, which is difficult to perform correctly. In fact, query containment and equivalence is NP-complete for conjunctive queries and undecidable for queries involving negation [27]. Instead of modeling queries in terms of syntactical constructs (e.g., *select*, *where* clauses), we propose to bypass the complexities and intricacies of syntax analysis and model queries in terms of data access, i.e., the actual database tuples that are returned as a result of query execution.

The examples above give the reader the insight into why syntax-based approaches may not perform well. We shall demonstrate this limitation experimentally in a later section,

and also experimentally compare it with our data-centric approach.

IV. User Profiles in Data-Centric Approach

The main premise of the data-centric approach to the database insider threat detection problem is: the actual data returned after query execution is the most important discriminator of user intent. In a logical sense, we care about the *semantics* of the queries, not their syntax. This notion is intuitive because a malicious insider typically tries to acquire, refine and enhance his/her knowledge about different data points and their relationships – this act likely involves data access patterns that may be atypical for his/her job function. The deviation from normal access patterns should occur in both the data harvesting type, the masquerading type of attacks, and also the compromised account case (where an intruder gains access to an insider's account).

To develop a suitable data-centric query representation format, we begin by considering the *Universal Relation* [17]:

A. Database Schema and the Universal Relation

A relational database [1] may consist of multiple relations with attributes and relationships specified by multiple *primary key* and *foreign key* constraints. One way of visualizing such a database is as a single relation, called the *Universal Relation* [17], incorporating the attribute information from all the relations in the database. Our goal is to be able to keep track of user access patterns to data tuples in the database.

A straightforward approach to profiling user data access might proceed as follows – for a universal relation with n attributes, each data tuple is viewed as a point in some n dimensional space; a user may be profiled by the set of such points he/she typically accesses. Each query is thus a set of points in this space. However, in practice, the number of attributes n in the universal relation and especially the number of data points accessed by users are prohibitively large; this straightforward method is unlikely to be scalable. Moreover, it is not clear how a new point set representing the new query can be classified as normal/abnormal using this method.

Our approach is as follows: instead of keeping track of individual data tuples, we compute a statistical "summary" of the query's result tuples. The summary for a query is represented by a vector of fixed dimension regardless of how large the query's result tuple set is. This way, past queries (i.e. normal queries) from a user can be intuitively thought of as a "cluster" in some high dimensional space. We have to emphasize that clustering is only one of several statistical learning technique we will test for this problem. The term clustering is used here to give the reader an intuitive sense of the model. When a new query comes, if it "belongs" to the user's cluster, it will be classified as normal, and abnormal otherwise.

Relations consist of various kinds of numeric and non-numeric attributes (columns) and data tuples (rows). The execution of a query results in the return of a subset of data tuples from the database as a result of database operations such as *projection*, *selection* and *join*. Identification of the schema for the database queried allows us to represent each row (tuple) in the execution result of a query as a tuple in the universal relation corresponding to the database. A data-centric query representation format called an *S-Vector* (statistics/summary vector) is described in the following subsection.

B. S-Vectors

An S-Vector is a multivariate vector composed of real-valued features, each representing a statistical measurement; it is defined by the columns of the universal relation corresponding to a database. Each attribute of the universal relation contributes a number of features to the S-Vector:

- Numeric Attributes: Each numeric attribute contributes the measurements *Min* (minimum value), *Max* (maximum value), *Mean*, *Median* and *Standard deviation*.
- Non-Numeric Attributes: The statistics computation does not make sense for non-numeric attributes such as *char* and *varchar*. For categorical attributes, one option is to expand a k valued attribute into k binary-valued numeric attributes (value 1 if the category is represented in the set of result tuples and 0 otherwise) and compute statistics on it as usual. However, the expansion of categorical attributes may result in an *S-vector* that has far too many dimensions – we compromise by replacing each categorical attribute with two numeric dimensions representing the *total count* of values, as well as the number of *distinct values* for this attribute in the query result.

The S-Vector format for a database is determined by its schema; the value of the S-Vector for a query is determined by executing the query and computing the relevant attribute statistics based on the set of result tuples and the result schema. Table I shows the S-Vector format for a database consisting of a single relation. To illustrate how an S-Vector value for a query is generated, consider the following query executed against the database in Table I:

```
SELECT p.cost
FROM PRODUCT p
WHERE p.type = 'abc';
```

For this query, the result schema consists of the single column Product.cost and statistics computed on the result tuples are used to populate the *Product.Min*, *Product.Max*, *Product.Mean*, *Product.StdDev* and *Product.Median* features of the S-Vector format for the database – the result is the S-Vector representation of this query.

V. A taxonomy of query anomalies from the data-centric view

In order to evaluate the effectiveness and accuracy of a threat detection engine, a taxonomy of query anomalies is useful to aid in reasoning about potential solutions. Subsequent experiments can be analyzed in the light of this taxonomy and the performance of detection schemes with respect to specific anomalies can be evaluated.

TABLE I

STATISTICS VECTOR FORMAT FOR SAMPLE DATABASE SCHEMA

Database Schema		S-Vector Features
Relation	**Attribute**	
Product	Product.type(varchar)	Product.type.ncount Product.type.ndistinct
	Product.cost(numeric)	Product.cost.Min Product.cost.Max Product.cost.Mean Product.cost.StdDev Product.cost.Median

We will classify query anomalies based on how "far" the anomalous query is from a normal query. From a data centric view point, two queries are represented by the two query execution results, each of which consists of the result schema (the columns) and the result tuples (the rows). If the result schemas are (very) different, the two queries are different. If the result schemas are similar, then we need to look into how different the result tuples are. On this basis we classify query anomalies.

A. Type 1 – Different Result Schema/Different Result Tuples

This is a typical case since queries that differ in the result schema have distinct SQL expressions (especially in the SELECT clause) and should be readily detected by both syntax-centric and data-centric schemes (since result tuples differ). From the insider threat perspective, data harvesting and masquerading can both result in this type of anomaly. As an example, consider the two queries to the database described in Table I:

```
Query 1: SELECT p.cost
FROM PRODUCT p
WHERE p.type = 'abc';
```

```
Query 2:  SELECT p.type
FROM PRODUCT p
WHERE p.cost < 1000;
```

Distinguishing these kinds of queries has received the most attention in the literature (e.g., [16]) especially in the context of masquerade detection and Role Based Access Control (RBAC), where different user roles are associated with different authorizations and privilege levels (to execute commands, queries etc.) [28]. An attempt by one user-role to execute a query associated with another role indicates anomalous behavior and a possible attempt at masquerade.

Syntax-based anomaly detection schemes have been shown to perform well for this case and it is our contention (and indeed we show later) that data-centric schemes should also be equally effective – different result schema necessarily implies different result tuples, and therefore different statistical characteristics for the results.

B. Type 2 – Similar Result Schema/Different Result Tuples

The result schema is similar for two queries in this case, but the data tuples are (significantly) different. We consider two sub-cases within this category – one is hopefully distinguished by most good syntax-centric schemes, the other is usually undetected by most syntax modeling schemes. As an example, consider the base query:

```
SELECT *
FROM PRODUCT p
WHERE p.cost == 1000;
```

Execution of this query results in the schema $p.type, p.cost$ and data corresponding to the WHERE condition $p.cost = 1000$. We consider two variations of this query:

- *Type 2(a) (Distinct Syntax)* – Consider the query:
  ```
  SELECT *
  FROM PRODUCT p
  WHERE p.cost < 1000 AND
        p.type = 'abc';
  ```
  This query has the same result schema as the previous one with a possibly different result tuple-set (matching the additional constraint of the product type); however, the SQL expression syntax is distinctly different, and the WHERE clause has an additional attribute that is checked ($p.type$) compared to the previous query. Good syntax based analysis schemes should be able to detect this variation.
- *Type 2(b) (Similar Syntax)* – Now consider another query:
  ```
  SELECT *
  FROM PRODUCT p
  where p.cost < 1000 AND
        p.cost > 1000;
  ```
  This query has the same result schema as the previous two; however the result tuples are the complement of that for the first query. Syntax analysis of the SQL expression would show the same attribute in the WHERE clause as the first; most syntax-centric modeling schemes (e.g., [16]) would have an identical representation for this query as the first. The query can be rewritten in multiple ways (e.g. *p.cost* != 1000) with various combinations of constants, arithmetic and logical operators; even a very detailed syntax-based modeling scheme may be hard-pressed to consider all variations. However, data-centric modeling schemes are expected to readily identify this variation since the statistical characteristics of the results are likely to be vastly different from those of the first.

From the insider threat perspective, data harvesting and masquerading can both result in this type of anomaly. Another example of a well-known attack class that may fall in this category is *SQL-injection* since a typical attack is one that injects input causing condition checks to be bypassed resulting in the output of all tuples – e.g., a successful exploit of the first example above may lead to the execution of

```
SELECT * FROM PRODUCT p
WHERE 1;
```

C. Type 3 – Similar Result Schema/Similar Result Tuples:

A query whose execution results in a (statistically) similar schema and result tuples as another is considered to be similar from a data-centric viewpoint. Clearly, if the queries have the same syntax, then their resulting schemas and tuples are the same and they *are* identical from both the data-centric and syntax-centric view. The interesting case arises when a query producing the same result as another differs in syntax – syntax-based detection schemes are designed to disallow these queries or automatically flag them as anomalous. However, the question of whether such a query is truly an anomaly or not requires further analysis. Two distinct sub-cases may be considered depending on query semantics:

- *Type 3(a) (Similar Semantics)* – In this case, denying the query is an unnecessary restriction that syntax-centric schemes impose on the user. As an example, consider the query:

  ```
  SELECT p.type
  FROM PRODUCT p
  where p.cost < 1000;
  ```

  and the alternate query:

  ```
  SELECT p.type
  FROM PRODUCT p
  WHERE p.cost < 1000 AND p.type IN
  (SELECT q.type FROM PRODUCT q);
  ```

  Here, the results of the queries are the same, and so are the query semantics and the underlying user intent – to retrieve product types that cost less than a certain amount, but the queries have very different SQL expression syntax. Data-centric approaches would correctly permit the second variation that may be denied by syntax-centric modeling schemes.
- *Type 3(b) (Different Semantics)* – Even though two queries expose exactly the same tuples, in some cases, an attacker may learn additional information because of their different semantics. As an example, consider this query in relation to the first above:

  ```
  SELECT p.type
  FROM PRODUCT p
  WHERE true;
  ```

  Now assume, for the sake of illustration, that the attacker is attempting to see all product types (data harvesting). If the above query returns more (or different tuples) with respect to the first example, then the data-centric approach should, conceptually detect this. But if the result tuples are exactly the same, this would (as expected) be permitted by the data-centric approach. However, the attacker has now gained the additional information (based on his results from the first query above), that all product types in the database cost less than 1000, and has refined his knowledge regarding some entity. This kind of successive knowledge accrual has received much interest in the areas of privacy preserving data mining and query auditing ([29], [30]). Syntax-centric schemes would blindly disallow this, but as we have shown, this may be an extreme step that may restrict many legitimate queries. The attack here arises from information refinement through temporal interaction between a user and a database and not from a property of the query itself (i.e., its syntax or result data). Exploiting temporal features from a data-centric viewpoint is an important future research direction of ours. It should be noted, however, that it is difficult for an attacker to intentionally exploit this condition, since he is unable to predict the nature of query output to ensure that result statistics are unchanged from a normal query. In any case, addressing this type of attacks is beyond the scope of this paper.

The different types of query anomalies are summarized in Table 2. In the next section, we present our experiments testing a prototype of a data-centric insider detection system and validate our approach for detecting types 1 and 2 anomalies. Type 3 is outside the scope of this paper.

VI. Experimental Validation

We begin this section with a description of a prototype of a data-centric anomaly detection system called *QStatProfiler*. We will also elaborate on the framework of the experimental setup, and detail our validation steps.

A. Test Setup

The testing environment consists of a web application for Graduate Student Admissions (*GradVote* that relies on a Postgresql [31] database at the back-end. There are a number of users of the system that interact with the database by means of queries – this interaction happens primarily via the web application. The users fall into several user categories, including *Chair*, *Faculty* and *Staff*.

The database schema consists of 20 relations with multiple (over 20 for some tables) numeric and non-numeric attributes and 39 multi-level views (i.e., the views refer to base relations as well as to other views). The training and testing dataset consists of tens of thousands of user queries that are labeled both by individual user-name as well as by user-role. These views are significantly complex, possessing multiple subqueries, complex joins and computed attributes (e.g, *sum*, *average* of the values in a numerical field).

Our system, *QStatProfiler* is positioned so that the interaction channel between the application and the database is visible (i.e., it can observe the queries to the database as well as the query execution results returned to the application). As queries are submitted to the database and result tuples are returned, the ID system simultaneously computes query statistics and the S-Vector for the query. The anomaly detection engine is flexible and can accommodate a variety of machine learning/clustering algorithms – we elaborate on different algorithms and anomaly detection goals in the following sub-section. The high-level setup is depicted in Figure 1 (configured to perform role-based masquerade detection) – the solid figures represent system

TABLE II
QUERY ANOMALIES – DATA-CENTRIC VIEW

Anomaly Cases (Result Schema/Tuples)	Types	Detected by Syntax-Centric?	Detected by Data-Centric?	Attack Models
Type 1. Diff. Schema/Diff. Results		Yes	Yes	Masquerade
Type 2. Similar Schema/Diff.Results	(a) Distinct Syntax	Yes	Yes	SQL-injection
	(b) Similar Syntax	No	Yes	Data-harvesting
Type 3. Similar Schema/Similar Results	(a) Diff. Syntax/Similar Semantics	Yes (false positive)	Yes (allowed)	Data harvesting
	(b) Diff. Syntax/Diff. Semantics	Yes	No	

components, the broken lines indicate the flow of information among these components. We elaborate further on various aspects of this system below.

Query Filtering: An important task for database intrusion detection is to construct accurate profiles for users/roles in order to perform anomaly detection effectively. For this purpose, it is necessary to overlook queries that are generated not by any specific user, but by the interface of the web application (thus these are common for all users). For example, the application may issue a query to the database to obtain the currently active list of users, or the time-line for a particular activity, and so on – these queries may sometimes be generated as part of application startup. The set of these queries is well-known *a priori*, since they may be embedded in the application code and can be overlooked for the purpose of user profiling. In our case, we maintain a list of *url* tags that indicate common application queries – queries generated by these pages are classified as *Framework Queries* by *QStatProfiler*.

Query Parsing and Unfolding: This component is concerned with obtaining the mapping between the schema of the result set and the overall schema of the database. The syntax of a user query may not refer directly to elements of the base database schema (i.e., base relations and their attributes). References may be made to views that might refer to other views; the use of aliases and in-line subquery definitions can complicate the task of schema mapping. *QStatProfiler* uses a query parsing component that is tailored to the *Postgresql* SQL syntax. Query parse trees are constructed which are then analyzed to determine the subset of the database relations and attributes that are present in the result tuples. The output of this phase is thus a set of relations and attributes that describe the result tuples.

B. Performance Considerations – S-Vector Approximation

Our approach relies on actual execution of SQL queries and analysis of results to enable anomaly detection. This may lead to concerns regarding the performance penalty of the approach, especially with regard to database performance overheads. We address these concerns in this subsection.

First, we argue that the approach does not impose significant *additional* burden to the database server. In most application environments (e.g., web database applications), execution of database queries is part of typical application function. For example, a user might submit queries through a web form; the queries are executed at a remote database server and the results are made available to the application. Our system (Figure 1) operates as a passive component between the application and the database server, observing queries and the corresponding results without disrupting normal functioning. The database does not experience any additional load due to the anomaly detection system; the computational cost of calculating result statistics falls on a different host that runs the ID system (QStatProfiler).

Secondly, and as alluded to earlier, the data-centric approach needs to see some data, necessitating some performance penalty if we compare it to the syntax-centric approach on a malicious query that the syntax-centric approach is able to detect. However, as we shall see, the execution of one pinelined round in the RDBMS is sufficient for the data-centric engine to perform well. The extra burden put on the server is minimal. In general, we propose to utilize only k tuples from the result set to build the corresponding S-vector. Two variations are considered for S-Vector approximation of a query in the online (testing) phase:

Top–k tuples: In this case, only the top (initial)-k tuples in the result set are considered to approximate the entire result set. Statistics computed from these tuples are used to generate the *S-Vector* representation of the query.

Random–k tuples: k number of tuples are chosen at random from the complete result set and considered for computation of the statistics needed for the S-Vector representation. This approach is expected to produce better accuracy as compared to the top-k approach as it is not likely to be sensitive to specific orderings of the result tuples by the database (this is especially important if the SQL query contains 'ORDER BY' clauses). Fortunately, we shall show that our particular way of picking the distance function seems to **not** be very sensitive to result set ordering.

In the next subsection, we consider the validation of the data-centric approach from two angles – detecting anomalies that fall into Type 1 and Type 2 in table II. The two cases are considered separately.

C. Validation – Type I Anomalies

The typical case of query anomaly detection involves detecting instances of Type 1. Although we claim conceptually that the data-centric approach should prove effective, we have to confirm this intuition with real experiments. We consider the specific case of role-based masquerade detection as the test setting – where specific queries are associated with user roles and execution of a query by a user belonging to another group constitutes an anomaly. We will benchmark this aspect of

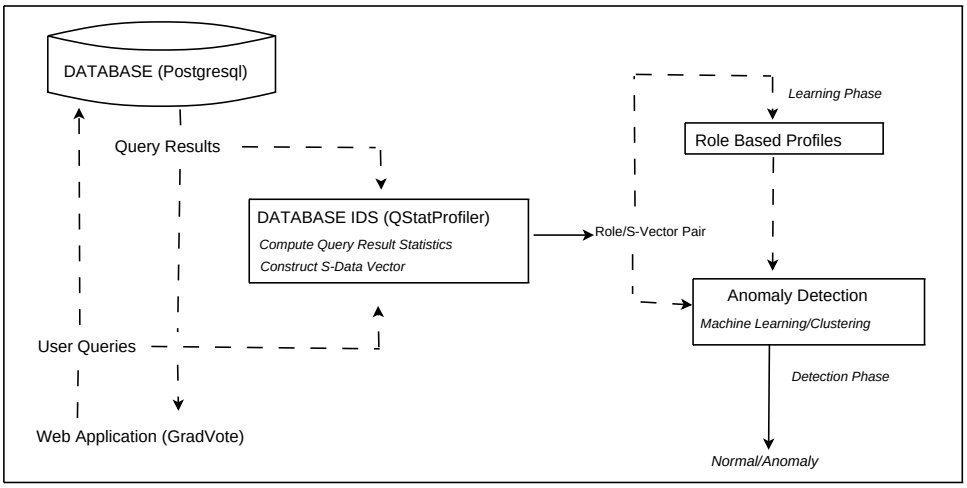

Fig. 1. Design of intrusion detection system for detecting insider attacks against databases

TABLE III
DETECTION PERFORMANCE – TYPE 1 ANOMALIES (ROLE MASQUERADE)

Roles	Algorithm	C quip.	M quip.	F quip.	S-V (all)	S-V I(20)	S-V R(20)	S-V I(10)	S-V R(10)	S-V I(5)	S-V R(5)
Chair Vs Faculty	N-Bayes	81.67%	85.33%	75%	85%	85%	82.67%	78.33%	77%	81.67%	**90%**
	Dec. Tree	**88%**	**87.67%**	**87.67%**	**96.33%**	**88.3%**	**88.3 %**	**89%**	**88.67%**	**88.67%**	88.67%
	SVM	83.3%	81%	**87.67%**	82.33%	74.67%	77%	71.33%	75.67%	68%	74.33%
Chair Vs Staff	N-Bayes	58%	93.5%	95.5%	60.5%	59%	60.5%	62%	57.5%	62.5%	60.5%
	Dec. Tree	**75%**	**88%**	**96%**	**95.5%**	**92.5%**	**96%**	**96%**	**93%**	**95%**	**92.5%**
	SVM	51.5%	84.5%	**96%**	80%	84%	85.5%	78.5%	81.5%	85.5%	82%
Faculty Vs Staff	N-Bayes	84.33%	90.67%	93%	58.67%	61.3%	60.3%	60.3%	59.3%	63%	60%
	Dec. Tree	**90%**	**93.67%**	**95.67%**	**89.3%**	**92.3%**	**91.67%**	**92%**	**93.67%**	**91.33%**	**91.67%**
	SVM	87%	93%	**95.67%**	69.67%	71.67%	71%	69.33%	72%	68.67%	72%

performance by comparison with a syntax-centric scheme that has been successfully applied to the detection of role-based anomalies [16]. Our goal here is to show that our data-centric scheme performs at least as well as syntax-centric schemes for Type 1 anomalies.

Syntax-Centric Format: For the sake of completeness, we present a brief summary of the syntax-centric data formats presented in [16]. Three representations are considered – *Crude (C-quiplet)*, *Medium (M-quiplet)* and *Fine (F-quiplet)* recording varying levels of detail.

- C-quiplet: This is a *coarse-grained* representation consisting of the SQL-command, count of projected relations, count of projected attributes, count of selected relations and a count of selected attributes.
- M-quiplet: This *medium-grained* format records the SQL command, a binary vector of relations included in the projection clause, an integer vector denoting the number of projected attributes from each relation, a binary vector of relations included in the selection clause, and an integer vector counting the number of selected attributes from each relation.
- F-quiplet: This is a fine-grained query representation. It differs from the M-quiplet in that instead of a count of attributes in each relation for the selection and projection clauses, a binary value is used to explicitly indicate the presence or absence of each attribute in a relation in the corresponding clauses.

Test Procedure: The available dataset of queries is labeled by the roles *Staff*, *Faculty*, and *Chair*, in addition to *Framework*, for the common application-generated queries, as described above. The set of queries is randomized and separated into *Train* and *Test* datasets of 1000 and 300 queries respectively. For benchmarking performance, four query data representations are tested – our *S-Vector* (dimensionality – 1638) and the syntax-centric *C-quiplet* (dimensionality – 5), *M-quiplet* (dimensionality –73) and *F-quiplet* (dimensionality – 1187) data representations from [16]. Three machine learning algorithms are tested with each of these data formats – Naive Bayes Classifier (NBC), Decision Tree Classifier and Support Vector Machines. Since role information is typically available in a masquerade detection environment, the algorithms are trained and tested on labeled data (supervised learning). These well-known approaches are described briefly.

- *Naive Bayes Classifier (NBC):* The NBC [32] is a well-known technique that has proven to be effective in many applications such as text classification. The classifier is based on *Bayes Theorem* and operates under the *Maximum Aposteriori Probability* (MAP) decision rule – given

an instance to be classified, the classifier decides on the correct class if it is more probable than any other class. If the attributes of this instance are $(a_1, a_2, a_3, ...a_n)$ and C is the set of classes, the most probable class C_{MAP} is given by:
$C_{MAP} = arg\ max_{c_j \in C}\ P(c_j|a_1, a_2, a_3, ..., a_n)$
By *Bayes Theorem* and by the conditional independence attribute that NBC makes with respect to attributes, this reduces to choosing C_{MAP} as
$C_{MAP} = arg\ max_{c_j \in C}\ P(c_j)\Pi_i\ P(a_i|c_j)$
The probabilities (including conditional probabilities) are estimated based on the training data. Further details on NBC are available in [32].

- *Decision Trees:* Decision tree algorithms are well known machine learning techniques that are popular in data mining applications. A decision tree is a structured plan to test attributes in order to eventually arrive at a class prediction – this structure is similar to a tree (hence the name). The attributes to be tested are in the order of the *information gain* [32] that they possess – at each step of tree construction, the attribute with the highest information gain (among those not yet part of the tree) is added to the decision tree. The tree gets constructed during the training phase as is dependent on the attribute characteristics of training instances. We utilize the J.48 decision tree algorithm [33] in our experiments.
- *Support Vector Machines:* Support vector machines perform classification by constructing hyperplanes for various classes and a decision boundary for class separation. The decision boundary is constructed during the training phase and is used during the online (testing) phase to classify new instances. The training instances that lie on the hyperplanes that lead to the definition of the decision boundary (i.e., those that would change the solution if they were omitted) are called *support vectors*. Details on Support Vector machines are available in [34].

The results for the binary classifiers for masquerade detection are depicted in Table III (the best performance for each format with respect to separating user roles is shown in boldface). In the table, I(k) and R(k) denote the Initial-k and Random-k S-Vector approximations. We note that the performance of the S-Vector based detection is comparable to those of the syntax-based schemes (even better in some cases). We also note that the Top-k and Random-k S-Vector approximations perform competently. We note that for the *Faculty Vs Staff* case, the syntax centric *F-quiplet* performs better than the S-Vector (95.67% to 89%). Our analysis shows that the *Faculty* and *Staff* roles are not well separated in terms of data access (this also explains the poor performance of the Naive-Bayes and SVM algorithms for the S-Vector in this case) which may be responsible for this effect. It is seen that the Top-k and Random-k approximations give slightly better results than the full S-Vector – this may be due to the effects of overfitting for this particular dataset. However, in general, the techniques show comparable performance, and the Decision-Tree algorithm is found to work best with the S-Vector representation.

D. Validation – Type 2 Anomalies

The focus here is on detecting queries are similar in syntax, but differ in output data (data-values, output volume, or both). This is a significant query anomaly since, in a typical attack, a minor variation of a legitimate query can output a large volume of data to the attacker. This may go undetected and may be exploited for the purpose of data-harvesting. In other attack variations, the volume of the output may be typical, but the data values may be sensitive. These kinds of attacks fall into Type 2(b) in Table II. Since Type 2(a) anomalies are detected easily by syntax analysis, this is similar to Type 1 for testing purposes, and will not be considered separately.

Test Procedure: Since a suitable dataset of Type 2(b) anomalies was not readily available to us for testing purposes (since these may be considered possibly malicious or unusual in typical domains), we manually create a test-set. The set consisted of variations of normal queries (i.e., queries normally executed by users in our dataset) that were designed to bypass syntax-centric schemes; this 'anomaly set' has approximately the same distribution of distinct queries as the original dataset. The query variations are easy to generate by varying arithmetic and logical operators and constants. As an example, consider the query:

```
SELECT * FROM vApplicants
WHERE reviewStatusID = 'a'
AND reviewStatusID = 'b';
```

A suitable Type 2(b) variation is as follows:

```
SELECT * FROM vApplicants
WHERE reviewStatusID = 'a'
OR reviewStatusID = 'b';
```

It must be noted that the queries considered here are different from masquerade attacks (since they are not representative of any authorized user of the system) and are typically not available for training a detection system. Hence, supervised anomaly detection approaches are not suitable here. We consider two detection techniques that detect potential anomalies based on a single class of *normal* queries – a *Cluster-Based Outlier Detection* method, and another approach we call *ATTRIB-DEVIATION* which is found to perform better.

Cluster-based Outlier Detection: The set of queries encountered during the training phase can be considered as points in an m dimensional vector (m is the dimensionality of the S-vector) space. A clustering technique (e.g., *K-means* clustering) can be used to discover clusters representing similar users (e.g., belonging to the same role) if role-information is missing; however, in our case role data is available and this information can be used to partition the data into role-clusters (i.e., the *Faculty* and *Staff* clusters). Additionally, the training data can be considered to be free from 'attack' (or bad) queries; if necessary, the training data can be made robust to the effects of bad queries or noise by pruning 5–10% of the

TABLE IV
TYPE 2(B) ANOMALIES – CLUSTER BASED OUTLIER DETECTION

Format	S-V (all)	S-V I(20)	S-V R(20)	S-V I(10)	S-V R(10)	S-V I(5)	S-V R(5)
Detection	83.87%	12%	67.7%	6.4%	45.1%	6.4%	35.4%

TABLE V
TYPE 2(B) ANOMALY DETECTION – ATTRIB-DEVIATION

Format	S-V (all)	S-V I(20)	S-V R(20)	S-V I(10)	S-V R(10)	S-V I(5)	S-V R(5)
Detection	90.3%	90.3%	90.3%	90.3%	90.3%	90.3%	90.3%
Min. No. Anom. Dimensions	5	4	5	4	5	4	5

outliers. Training data consisting of n queries (vectors) can be represented by the $n \times m$ matrix

$$A_{n\times m} = \begin{pmatrix} x_{11} & x_{12} & ... & x_{1m} \\ x_{21} & x_{22} & ... & x_{2m} \\ ... & ... & ... & ... \\ x_{n1} & x_{n2} & ... & x_{nm} \end{pmatrix}$$

For each user cluster, we select a point in the Euclidean space that is representative of the entire cluster, called the *cluster centroid*. A cluster centroid may be chosen so as to minimize the sum of the squared Euclidean distances of the cluster points:

$Centroid = \frac{1}{n}\sum_{i=1}^{n} x_i$

For a test S-Vector, the distances (using the Euclidean distance metric) from the cluster centroids are computed; the query is flagged as an outlier if the vector distance is greater than a specified threshold from any user. In our case, we specify the threshold as 3 times the standard deviation; this is the outlier specification typically utilized for a normal population. A typical performance result with two user clusters (*Chair* and *Faculty*) and corresponding anomalous query set is shown in Table IV.

Results with role-cluster based outlier detection provide some interesting insights. While detection rate for the S–Vector(all results) is reasonable (83.87%), it is seen that results with the Top-k results-approximation suffers significantly. On analysis, we find that many of the user queries make extensive use of the SQL *ORDER-BY* clause. Understandably, this results in a skewed representation of the overall result statistics in the top k tuples. This is ameliorated to some extent by the Random-k variation (e.g., for random $k = 20$, the detection rate improves to 67.7%); however, there is still a marked decline in performance indicating that the clustering scheme is sensitive to the approximation schemes and is affected negatively by them. Further analysis into the clustering reveals that this may perhaps not be the perfect scheme for anomaly detection. Although anomalies with significant variations in multiple dimensions are easily detected by clustering (as for many of our examples), this may not be true in the general case. Firstly, distances in high-dimensional space may be misleading indicators of anomalies because of the *curse of dimensionality*. For example, it is possible to have a highly anomalous value along a single dimension, which may not translate to a significant Euclidean cluster-distance (and *vice-versa*). As a final validation of the poor performance of clustering, we tested the original problem of masquerade detection (*Chair Vs Faculty*) using single-class outlier detection – the performance was below 10%.

We develop an alternate technique for anomaly detection that can deal with the shortcomings of clustering. This is based on an insight into the nature of query anomalies and how the corresponding *S-Vectors* are affected.

ATTRIB-DEVIATION: Consider, for example, that a user issues an anomalous query with a different statistic for the same attribute in the result schema as a normal query. In our representation, this difference shows up in one or more (depending on whether the attribute is *categoric* or *numeric*) dimensions of the S-Vector. Hence, monitoring for anomalies on *per-dimension* basis is a promising approach. Further, if a query generates unusual output for more than one attribute, this is likely to reflect in anomalous values for several S-Vector dimensions; thus, the number of anomalous dimensions for the S-Vector is a parameter that can be used for ranking potential query anomalies (i.e., queries with more anomalous S-Vector dimensions rank high as likely candidates for possible attacks). We utilize this approach for testing the custom-developed anomaly set – normal *Chair* and *Faculty* queries are used to compute the mean values of S-Vector attributes; three times the *standard-deviation* is again used as an anomaly separator. The results are summarized in Table V (the third row indicates the minimum number of anomalous dimensions for any query in the test-set that is found to be anomalous).

Again, the results reveal some interesting insights. While detection performance for the S-Vector (all results) improves as compared to the clustering approach (90.3% compared to 83.87%), we find that the attribute-deviation based scheme is remarkably resilient to the approximation schemes (both Top-k as well as Random-k). The approximation schemes perform as well as the full vector representation, but the Top-

k performs unexpectedly well even with queries generating specific ordering of results.

Based on our analysis, we offer the following explanation. First, note that a single anomalous attribute in the result corresponds to variations in multiple dimensions of the S-Vector, each of which represents a statistical measurement. Also the extent of the anomaly may vary between result attributes (e.g., some attributes may have more atypical values). While a selective ordering (e.g., by SQL *ORDER-BY* clauses) may offer a skewed view of overall result statistics, the *ATTRIB-DEVIATION* technique operates on a per-attribute basis and is thus still able to identify anomalies. Secondly, many queries have more than one anomalous attribute; hence selective ordering may mask anomalies in some attributes, but not in others. Thirdly, the selective ordering may not affect all statistical measurements of a single attribute equally (e.g., it may affect *Max*, but not *Median*) In our tests, we notice that while the detection performance of both the Top-k and the Random-k schemes remain the same, the ranking of query anomalies and the associated number of anomalous dimensions vary (typically fewer number of anomaly dimensions for Top-k, as expected); this is shown in the last row of Table V.

As a final validation of this detector, we test the problem of masquerade detection for (*Chair Vs Faculty*) (using the classes alternately as 'normal'). The detection performance with *Chair* (*normal*) was 94.7% and that with *Faculty* (*normal*) was 97.5%, slightly better than even the ML algorithms in Table III.

We believe that the good performance of the Top-k approximation with this detection technique has several practical implications. First, it indicates that a fast online anomaly detector can perform well by considering just a few initial output tuples. Randomized sampling of query results may not be feasible in all cases, especially for queries generating hundreds or thousands of output tuples (e.g., due to performance constraints), but our results here indicate that accuracy may not have to be sacrificed in the process of giving up random sampling. Further, we also believe that the S-Vector representation scheme and attribute-deviation based anomaly detection algorithm are quite resilient to attacks designed to mislead or bypass detection – we argue that it is very difficult for an attacker to craft queries so that multiple statistical measurements are controlled – a theoretical result may be an interesting research problem.

VII. Concluding Remarks

In this section we discuss some overall aspects of our solution, their practical implications and identify key goals for future research.

Queries: In order to characterize query results using S-Vectors, we need to express the schema of each query result in terms of the attributes of the base relations (*base schema*). For select-project-join (SPJ) queries on base relations, the base schema is easily determined. When SPJ queries are also expressed on top of views, then we employed the view unfolding technique [35] to determine the base schema. View unfolding recursively replaces references to a view in a query expression with its corresponding view definition. For a class of queries larger than SPJ queries on base relations and views, it is not clear if the base schema can be determined. For example, union queries can map two different attributes in base relations into a single one in the query result, as the following example shows:

```
SELECT g.name, g.gpa FROM GRADS g
  UNION
SELECT u.name, u.gpa FROM UGRADS u;
```

In this case, there is no dimension in the S-vector to accommodate the first attribute of the query result. The same is true for computed attributes in results of complex (aggregation, group-by) queries. To accommodate such cases, we plan to investigate data provenance techniques [36] and revise the definition and the use of the S-vector accordingly.

Databases: The framework proposed in this paper assumes that the underlying database is *static*, i.e., there are no updates. Although this assumption is true or adequate for a certain class of databases (e.g., in applications such as census reporting), we plan to extend our work to *dynamic* databases. The first challenge in this case is to determine if and when updates shift the boundary between normal and abnormal queries with respect to the initial database state. If the database instance is updated significantly, then our training sets and classifiers become obsolete. Two directions we plan to investigate are: (a) detect when a re-training of the system is needed, and (b) whether detecting abnormal activity using stable versions of the database is effective for periods between re-training.

Activity Context: In our approach, the context of a user's activity is a set of query results generated in the past by the same user or the group in which she belongs. When a user generates a new query result, then its S-vector is compared to the S-vectors representing the query results in the past. We plan to investigate richer activity contexts and examine their effectiveness in detecting sophisticated attacks. Such contexts might include statistics of a user's session with the database, temporal patterns of the query results generated by the user in the past (large results during tax season, many results during the holiday season) and so on.

Performance: In cases where user queries return a significantly large number of results, computing statistics over the entire query result for anomaly detection might be unacceptable from a performance standpoint. The top-k approximation proposed in Section VI improves performance without sacrificing accuracy. One potential drawback of this approach is that the queries in the training set might sort the results by a different attribute or in different order (ascending, descending) than an otherwise normal user query, thus leading to false positives. A possible solution to this problem is to choose one attribute of each base relation as the default order by attribute. Then, for every query in the training set add a *designated* ORDER BY clause that orders the result by the chosen attribute of the first base relation (alphabetically) used in the query. When a user query is submitted, the system

submits an additional query with the *designated* ORDER BY clause and uses this query result for detection.

Although random-k does not outperform top-k in our experiments, we expect random-k to perform consistently for a wider range of datasets and queries. Of course, a problem that arises then is how to sample a query result without computing the complete result, given that RDBMSs follow the pipelined query execution model. For this hard problem, we plan to leverage prior work on both SPJ queries [37], [38] and queries for data analytics in the area of approximate query answering [39]–[41].

In conclusion, the techniques that we have presented and analyzed in this paper show significant potential as practical solutions for anomaly detection and insider threat mitigation in database systems. Some open research issues still remain – we aim to develop and present efficient and practical solutions to these in future work.

REFERENCES

[1] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*. Reading, MA: Addison-Wesley, 2006.

[2] (2008) Open web application security project. [Online]. Available: http://www.owasp.org/

[3] (2007) Owasp top 10 2007. [Online]. Available: http://www.owasp.org/index.php/Top_10_2007

[4] (2008) Owasp-sql injection prevention cheat sheet. [Online]. Available: http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

[5] B. Schneier, *Secrets and Lies: Digital Security in a Networked World*. New York, NY: John Wiley and Sons, 2000.

[6] M. Bishop, "The insider problem revisited," in *Proc. of the 2005 Workshop on New Security Paradigms (NSPW'05)*, 2005, pp. 75–76.

[7] CSO Magazine, U.S. Secret Service, CERT, and Microsoft. (2007) 2007 E-Crime Watch Survey. [Online]. Available: http://www.sei.cmu.edu/about/press/releases/2007ecrime.html

[8] D. Cappelli. (2005) Preventing insider sabotage: Lessons learned from actual attacks. [Online]. Available: http://www.cert.org/archive/pdf/InsiderThreatCSI.pdf

[9] R. Brackney and R. Anderson, *Understanding the Insider Threat: Proceedings of a March 2004 Workshop*. RAND Corp, 2004.

[10] M. Schonlau, W. DuMouchel, W. Ju, A. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," *Statistical Science*, vol. 16, no. 1, pp. 58–74, 2001.

[11] C. Y. Chung, M. Gertz, and K. Levitt, "Demids: a misuse detection system for database systems," in *Integrity and Internal Control Information Systems: Strategic Views on the Need for Control*. Norwell, MA: Kluwer Academic Publishers, 2000, pp. 159–178.

[12] S. Y. Lee, W. L. Low, and P. Y. Wong, "Learning fingerprints for a database intrusion detection system," in *Proc. of the 7th European Symposium on Research in Computer Security (ESORICS'02)*, 2002, pp. 264–280.

[13] Y. Hu and B. Panda, "Identification of malicious transactions in database systems," in *Proc. of the 7th International Database Engineering and Applications Symposium*, 2003, pp. 329–335.

[14] F. Valeur, D. Mutz, and G. Vigna, "A learning-based approach to the detection of sql attacks," in *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA '05)*, 2005, pp. 123–140.

[15] A. Spalka and J. Lehnhardt, "A comprehensive approach to anomaly detection in relational databases," in *DBSec*, 2005, pp. 207–221.

[16] A. Kamra, E. Terzi, and E. Bertino, "Detecting anomalous access patterns in relational databases," *The VLDB Journal*, vol. 17, no. 5, pp. 1063–1077, 2008.

[17] D. Maier, J. D. Ullman, and M. Y. Vardi, "On the foundations of the universal relation model," *ACM Trans. on Database Syst.*, vol. 9, no. 2, pp. 283–308, 1984.

[18] P. Liu, "Architectures for intrusion tolerant database systems," in *Proc. of the 18th Annual Computer Security Applications Conference (ACSAC '02)*, 2002, p. 311.

[19] S. Wenhui and D. Tan, "A novel intrusion detection system model for securing web-based database systems," in *Proc. of the 25th International Computer Software and Applications Conference on Invigorating Software Development (COMPSAC '01)*, 2001, p. 249.

[20] V. C. Lee, J. Stankovic, and S. H. Son, "Intrusion detection in real-time database systems via time signatures," in *Proc. of the Sixth IEEE Real Time Technology and Applications Symposium (RTAS'00)*, 2000, p. 124.

[21] C. Kruegel and G. Vigna, "Anomaly detection of web-based attacks," in *Proc. of the 10th ACM conference on Computers and Communications Security (CCS'03)*, 2003, pp. 251–261.

[22] A. Srivastava, S. Sural, and A. K. Majumdar, "Database intrusion detection using weighted sequence mining," *Journal of Computers*, vol. 1, no. 4, pp. 8–17, 2006.

[23] A. Roichman and E. Gudes, "Diweda – detecting intrusions in web databases," in *Proc. of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, 2008, pp. 313–329.

[24] J. Fonseca, M. Vieira, and H. Madeira, "Online detection of malicious data access using dbms auditing," in *Proc. of the 2008 ACM symposium on Applied Computing (SAC'08)*, 2008, pp. 1013–1020.

[25] Q. Yao, A. An, and X. Huang, "Finding and analyzing database user sessions," in *Proc. of Database Systems for Advanced Applications*, 2005, pp. 283–308.

[26] P. Ramasubramanian and A. Kannan, "Intelligent multi-agent based database hybrid intrusion prevention system," in *Proc. of the 8th East European Conference (ADBIS '04)*, 2004.

[27] D. Calvanese, G. D. Giacomo, and M. Lenzerini, "On the decidability of query containment under constraints," in *Proc. of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '98)*, 1998, pp. 149–158.

[28] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The nist model for role based access control," in *Proc. of the 5th ACM Workshop on Role Based Access Control*, 2000.

[29] R. Agrawal and R. Srikant, "Privacy-preserving data mining," in *Proc. of the ACM SIGMOD Conference on Management of Data (SIGMOD '00)*, 2000, pp. 439–450.

[30] K. Kenthapadi, N. Mishra, and K. Nissim, "Simulatable auditing," in *Proc. of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '05)*, 2005, pp. 118–127.

[31] (2008) Postgresql. [Online]. Available: http://www.postgresql.org/

[32] T. Mitchell, *Machine Learning*. New York, NY: McGraw-Hill, 1997.

[33] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. San Francisco, CA: Morgan Kauffman, 2005.

[34] C. J. C. Burgess, "A tutorial on support vector machines for pattern recognition," *Data Min. Kowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.

[35] M. Stonebraker, "Implementation of integrity constraints and views by query modification," in *SIGMOD Conference*, 1975, pp. 65–78.

[36] P. Buneman, S. Khanna, and W. C. Tan, "Why and where: A characterization of data provenance," in *ICDT*, 2001, pp. 316–330.

[37] F. Olken and D. Rotem, "Simple random sampling from relational databases," in *VLDB*, 1986, pp. 160–169.

[38] S. Chaudhuri, R. Motwani, and V. R. Narasayya, "On random sampling over joins," in *SIGMOD Conference*, 1999, pp. 263–274.

[39] P. J. Haas and J. M. Hellerstein, "Ripple joins for online aggregation," in *SIGMOD Conference*, 1999, pp. 287–298.

[40] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, "Join synopses for approximate query answering," in *SIGMOD Conference*, 1999, pp. 275–286.

[41] B. Babcock, S. Chaudhuri, and G. Das, "Dynamic sample selection for approximate query processing," in *SIGMOD Conference*, 2003, pp. 539–550.