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

Sunu Mathew*[1], Michalis Petropoulos[2], Hung Q. Ngo[2], and Shambhu Upadhyaya[2]

[1] Information Security,
Amazon.com Inc., Seattle WA 98104, USA,
smathew@amazon.com,
[2] Computer Science and Engineering,
University at Buffalo, Buffalo NY 14260, USA,
(mpertropo, hungngo, shambhu)@buffalo.edu

**Abstract.** The insider threat against database management systems is a dangerous security problem. Authorized users may abuse legitimate privileges to masquerade as another user or to maliciously harvest data. We propose a new direction to address the problem. We model users' access patterns by profiling the *data points* that users access, in contrast to analyzing the *query expressions* in prior approaches. Our data-centric approach is based on the key observation that query syntax alone is a poor discriminator of user intent, which is much better rendered by *what* is accessed. We present a feature-extraction method to model users' access patterns. Statistical learning algorithms are trained and tested using data from a real Graduate Admission database. Experimental results indicate that the technique is very effective, accurate, and is promising in complementing existing database security solutions. Practical performance issues are also addressed.

## 1 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) is the fundamental means of data organization, storage and access in most organizations, services, and applications. Naturally, the prevalence of RDBMS's led to the prevalence of security threats against RDBMS's. An intruder from the outside, for example, may be able to gain unauthorized access to data by sending carefully crafted queries to a back-end database of a Web application. This class of so-called *SQL injection* attacks are well-known and well-documented, yet still very dangerous [1]. They can be mitigated by adopting suitable safeguards, for example, by adopting defensive programming techniques and by using SQL *prepare* statements [2].

An *insider attack* against an RDBMS, however, is much more difficult to detect, and potentially much more dangerous [3–5]. According to the most recent

* Work done as a graduate student at the University at Buffalo.

U.S. Secret Service/CERT/Microsoft E-Crime report, insider attacks constitute 34% of all surveyed attacks (outsiders 37%, and the remaining 29% 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 [6].

More formally, the RAND workshop devoted to insider threats [7] 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." Examples of insider attacks include *masquerading* and *privilege abuse* which are well-known threats in the financial, corporate and military domains; attackers may *abuse legitimate privileges* to conduct *snooping* or *data-harvesting* [3] with malicious intent (e.g., espionage).

### 1.1 Main Ideas

By definition, 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, a good statistical profiler should be able to detect non-stealthy sabotage attacks, quick data harvesting attacks, or masquerading attack because the computing footprints of those actions should be significantly different from the day-to-day activities, from a statistical point of view.

The user profiling idea for insider threat detection in particular and anomaly detection in general is certainly not new (see, e.g., [8]). In the context of an RDBMS (or any problem requiring statistical profiling), the novelty is in the answers to two critical questions: (1) *what is a user profile (and how to construct it)?* and (2) *which machine-learning techniques and models should we adopt so that the profiles are* practically useful *for the detection problem?* By "useful" we mean some relevant classes of insider attacks can be detected to a good degree of accuracy. 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.

Prior studies (e.g., [9–14]) have led to the development of intrusion detection systems (IDS) that aimed to protect databases from attacks. Our contribution is complementary, and is specifically on analyzing users' interactions with an RDBMS. Other behavioral features useful in insider threat detection (location of the attacker, informational correlation between consecutive queries, and temporal features such as time between queries, duration of session, etc.) are beyond the scope of this paper, and are considered future work.

Perhaps the most natural user "profile" is the set of SQL queries a user daily issues to the database, or more generally some feature vectors representing past queries. Indeed, [14] relied on the SQL-expression syntax of queries to construct user profiles. This approach has the advantage that the query processing of the insider detection system is computationally light: a new query is analyzed by some statistical engine; only queries accepted by the engine are then issued to the database. 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. On the one hand, queries may differ widely in syntax yet produce the same "normal" (i.e., good) output, causing the syntax-based detection engine to generate false positives. On the other hand, syntactically similar queries may produce vastly different results, leading the syntax-based engine to generate false negatives.

Our main idea and also 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 – rather than *how* he expresses it, i.e. the SQL expressions. In other words, this data-centric approach values the *semantics* of the queries more than their *syntax*. When a malicious insider tries to acquire *new* knowledge about data points and their relationships, the data points accessed are necessarily different from the *old* accessed points. The deviation occurs in the data harvesting attacks, the masquerading attacks, and also the compromised account case where an intruder gains access to an insider's account.

## 1.2 Contributions

Our first contribution is the proposed data-centric viewpoint, which to the best of our knowledge has not been studied in the database security and the insider threat literature. Intuitively, the data-centric approach has the following advantage: for an insider to evade our system, he has to generate queries with result statistically similar to the ones he would have gotten *anyhow* with legitimate queries using his existing privileges, rendering the attempt at circumvention inconsequential. In contrast, in the syntax-based approach queries with similar syntax can give different results: the attacker may be able to craft a "good-looking" malicious query bypassing the syntax-based detection engine to access data he's not supposed to access. This point is validated in Sections 3, 5 and 6.

The second contribution is a method to extract a feature vector from the result set of a query, which is the core of our answer to question (1) above. The dimension of the feature vector is only dependent on the database schema, but independent from size of the database. In particular, a query's feature vector's dimension is independent of how large the result set of a query is. This bounded dimensionality also partially addresses scalability and performance concerns the acute reader might have had. Section 4 details the method.

The third contribution is to address the following potential performance problem: 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 huge "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 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 RDBMS's.

The fourth contribution, presented in Section 5, is a taxonomy of anomalous database access patterns, which is needed to systematically evaluate the accuracy of both the data-centric and the syntax-centric approaches.

The fifth contribution is a relatively extensive evaluation of several statistical learning algorithms using the data-centric approach. Specifically, for the masquerade detection problem on a real Graduate Admission data set, we found that $k$-means clustering works very well, with detection rates of around 95-99%. For data harvesting detection, we develop an outlier detection method based on attribute deviation (a sort of clustering using the $L_\infty$-norm) which performs well. Furthermore, this method is suitable when the features are only extracted from a constant number of tuples of the result set, thus making it practical.

In summary, though our results are derived in the limited context of insider threat detection within database security, this paper is a first step in exploring the larger potential of the data-centric approach in anomaly detection.

**Paper Outline** The rest of this paper is organized as follows. Section 2 surveys background and related work. Section 3 demonstrates the limitations of the syntax-based approach, thus motivating the data-centric approach introduced in Section 4. Section 5 gives a brief taxonomy of query anomalies facilitating the experiments presented in Section 6. We further discuss our solution, its implications, and future research directions in Section 7.

## 2 Related Work

IDSs with direct or indirect focus on databases have been presented in the literature [15, 16]. In [17], temporal properties of data are utilized for intrusion detection in applications such as real-time stock trading. Anomaly detection schemes dealing with SQL injection attacks in Web applications were studied in [18, 12]. SQL injection attacks are a specific kind of database query anomaly that is detected by our approach in a straightforward manner as we shall show.

Data correlation between transactions is used to aid anomaly detection in [11]. Similarly, in [19], dependency between database attributes is used to generate rules based on which malicious transactions are identified. The DEMIDS system [9] detects intrusions by building user profiles based on their working scopes which consist of feature/value pairs representing their activities. 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 [10], which is another syntax-based system where SQL statements are summarized as regular expressions which are considered to be “fingerprints” for legitimate transactions. Yet another syntax-based approach was considered in [20] for web databases, where fingerprints of all SQL statements that an application can generate are profiled. A binary vector with length equal to the number of fingerprints is used to build session profiles and aid in anomaly detection. This approach made assumptions such as restrictions on the number of distinct queries possible, and may complement our approach in cases where the assumptions are valid. In [21], 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 [22]: 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 [23]; 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 most resemble ours are [13] and [14]. The solution in [13] 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 recent syntax-based work in [14] has the same overall detection goals as our work: detection of anomalies in database access by means of user queries. A primary focus on this paper will be on exposing the limitations of syntax based detection schemes; the approach in [14] will be used in this paper as a benchmark for evaluating the performance of our approach.

## 3 Limitations of Syntax-Centric Approach

This section demonstrates that two syntactically similar queries may generate vastly different results, and two syntactically distinct queries may give similar results. Consequently, SQL expressions are poor discriminators of users' intents. For example, a syntax-based approach may model a query with a frequency vector, each of whose coordinates counts the number of occurrences (or marks the presence) of some keywords or mathematical operators [14].

Consider the following query:

```
SELECT p.product_name, p.product_id
FROM PRODUCT p
WHERE p.cost = 100 AND p.weight > 80;
```

A syntactical analysis of this query and subsequent feature extraction (e.g., [14]) 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 AND p.weight = 80;
```

This query has the same syntax-based feature set as the previous one; however, the data tuples accessed in the two cases are vastly different.

Conversely, suppose we rewrite the first query as follows:

```
SELECT p.product_name, p.product_id
FROM PRODUCT p
WHERE p.cost = 100 AND p.weight > 80
AND p.product_name IS NOT NULL;
```

This query is syntactically different (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 anomalous with respect to the first.

Syntax analysis, even if very detailed (taking into account differences in the operand difference between '=' and '>' in the above examples) is complicated given the expressiveness 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 [24]. Our data-centric approach bypasses the complexities and intricacies of syntax analysis.

## 4 Data-Centric User Profiles

A relational database often consists of multiple relations with attributes and relationships specified by multiple *primary key* and *foreign key* constraints. One can visualize a database as a single relation, called the *Universal Relation* [25], incorporating the attribute information from all the relations in the database.

Our approach profiles users as follows: for each query 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'd like to emphasize that clustering is only one of several statistical learning technique we will adopt 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.

Our query summary vector is called an *S-Vector*. 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 according to the following rules.

**Table 1.** Statistics Vector Format for Sample Database Schema

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

*Numeric Attributes:* each numeric attribute contributes the measurements *Min* (value), *Max* (value), *Mean*, *Median* and *Standard deviation*.

*Non-Numeric Attributes:* the standard statistics do not make sense for non-numeric attributes (e.g., *char* and *varchar*). For categorical attributes, one option is to expand a $k$-value attribute into $k$ binary-valued attributes (value 1 if the category is represented in the set of result tuples and 0 otherwise) and compute statistics on them as usual. However, the expansion of categorical attributes may result in an *S-vector* that has far too many dimensions, affecting the time-performance of the learner. 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. Table 1 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 1:

```
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.

## 5 A Data-Centric Taxonomy of Query Anomalies

In order to evaluate the effectiveness and accuracy of a threat detection engine, a taxonomy of query anomalies can help us reason about potential solutions. Subsequent experiments can analyze the performance of detection schemes with respect to specific anomalies in light of this taxonomy. We shall 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 result sets, 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. Table 2 provides the taxonomy summary.

**Table 2.** A Taxonomy of Query Anomalies

| **Anomaly Cases** | **Types** | **Detected by Syntax-Centric?** | **Detected by Data-Centric?** | **Attack Models** |
|---|---|---|---|---|
| **Type 1** Different Schema/ Different Results | | Yes | Yes | Masquerade |
| **Type 2** Similar Schema/ Different Results | (a) Distinct Syntax | Yes | Yes | SQL-Injection Data-Harvesting |
| | (b) Similar Syntax | No | Yes | |
| **Type 3** Similar Schema/ Similar Results | (a) Different Syntax/ Similar Semantics | False Positive | Yes (True Positive) | Data Harvesting |
| | (b) Different Syntax/ Different Semantics | Yes | No (Rare) | |

**Type 1: Distinct Schema and Tuples** Anomalous queries of this type have result sets whose columns *and* rows are very different from those of normal queries. Intuitively, anomalous queries of this type should be detected by both the syntax-based and the data-centric approaches. The syntax-based approach works because queries that differ in the result schema should have distinct SQL expressions (especially in the SELECT clause). The data-centric approach works because the S-vector of the anomalous query not only differ in the dimensions (the result schema) but also in the magnitudes in each dimension (the statistics of the result tuples). From the insider threat perspective, data harvesting and masquerading can both result in this type of anomaly. As an example, consider the following two queries to the database described in Table 1:

```
Query 1: SELECT p.cost                Query 2: SELECT p.type
         FROM PRODUCT p                        FROM PRODUCT p
         WHERE p.type = 'abc';                 WHERE p.cost < 100;
```

Distinguishing these kinds of queries has received the most attention in the literature (e.g., [14]) especially in the context of masquerade detection and Role Based Access Control (RBAC) [26], where different user roles are associated with different authorizations and privilege levels. 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 we experimentally show later that data-centric schemes are also equally effective.

**Type 2: Similar Schema, Distinct Tuples** Anomalous queries of this type have result sets whose columns are similar to normal queries, but whose rows are statistically different. The syntax of type-2 anomalous queries might be similar to or different from normal queries. For example, consider the following normal query:

```
SELECT *
FROM PRODUCT p
WHERE p.cost = 100;
```

Execution of this query results in the schema `(p.type, p.cost)` and data corresponding to the `WHERE` condition `p.cost = 100`. On the one hand, the following type-2 anomalous query has the same result schema as the normal one with a statistically different result tuple-set (matching the additional constraint of the product type):

```
SELECT *
FROM PRODUCT p
WHERE p.cost < 100 AND p.type = ‘abc‘;
```

The SQL expression syntax is also distinctly different from the normal query. The WHERE clause has an additional attribute that is checked (`p.type`) compared to the previous query. On the other hand, the following type-2 anomalous query has the same result schema and also similar syntax as the normal query:

```
SELECT *
FROM PRODUCT p
WHERE p.cost < 100 AND p.cost > 100;
```

Yet the result tuples are the *complement* of that of the normal query. Thus, we further classify type-2 anomalous queries into type-2a and type-2b, where type-2a contains type-2 anomalous queries whose syntax are also distinct from normal ones, and type-2b contains the rest. The intuition is that a syntax-based scheme such as that in [14] is unlikely to be able to detect type-2b anomalous queries. Indeed, the scheme in [14] represents the above type-2b example query and the normal example query identically. Furthermore, type-2b anomalous queries can be rewritten in multiple ways (e.g. `p.cost != 100`), varying combinations of constants, arithmetic and logical operators; even very detailed syntax-based models may be hard-pressed to consider all variations. We will show that data-centric schemes are likely able to detect both of these anomalous types.

From the insider threat perspective, data harvesting and masquerading can both result in type-2 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;
```

**Type 3: Similar Schema and Tuples** A query whose execution results in a similar schema and tuples as a normal one is considered to be similar from a data-centric viewpoint. Clearly, if the queries also 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 a normal query is syntactically different from the normal query. The question is, should we consider such a query "anomalous"?

On the one hand, it seems to be obvious that a user accessing the same data schema and tuples as those in his normal access patterns should not be flagged as malicious regardless of how different the syntax of the queries he issued. For example, the following two queries should be considered identical:

```
SELECT p.type                  SELECT p.type
FROM PRODUCT p                 FROM PRODUCT p
WHERE p.cost < 100;            WHERE p.cost < 100 AND p.type IN (
                                  SELECT q.type
                                  FROM PRODUCT q
                               );
```

The two queries have identical outputs, semantics, and thus user intent. We will refer to an "anomalous" query of this type a *type-3a query*. Note again that "anomalous" is not the same as "malicious." Our approach will not raise a red flag, while the syntax-based approach would issue a false positive.

On the other hand, two queries resulting in the same output might actually reveal *more* information than what is in the output. To see this, we have to look a little deeper into the *semantics* of the queries. Consider the following query in relation to the ones from the previous paragraph.

```
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 in the rare case when 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 query from the previous paragraph), that all product types in the database cost less than 100, and has refined his knowledge regarding some entity. We call to this type of anomalous query *type-3b*.

This kind of successive knowledge accrual has received much interest in the areas of privacy preserving data mining and query auditing ([27, 28]). 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 presumably 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.

## 6 Experimental Validation

### 6.1 The Test Environment

The test environment consists of a real and currently active web application for Graduate Student Admissions (called *GradVote*) that relies on a Postgresql database at the back-end. Users of the system query the database primarily via the web application. The users fall into several roles, 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 datasets consist of tens of thousands user queries labeled both by individual username as well as by user-role. These views are significantly complex, possessing multiple subqueries, complex joins and statistical attributes.

Our system, *QStatProfiler*, is positioned in the middle of the interaction channel between the application and the database. It observes the queries to the database as well as the results returned to the application. As queries are submitted to the database and result tuples are returned, QStatProfiler simultaneously computes query statistics and the S-vectors for the queries. QStatProfiler is flexible and can accommodate a variety of machine learning/clustering algorithms. We shall elaborate on different algorithms and anomaly detection goals later.

*Query Filtering:* The first task of *QStatProfiler* is profiling users or roles. It is thus necessary to ignore queries that are 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. This set queries is well-known *a priori*, since they may be embedded in the application code and can be ignored while profiling. In our case, we maintain a list of *url* tags that indicate common application queries, called *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 and 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, from which S-vectors are constructed.

### 6.2 Approximating S-vectors

As alluded to earlier, having to execute a query before classifying it as anomalous is a legitimate performance concern, which is addressed in this section.

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 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).

Second, 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 (a true positive!). 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, and is only marginally worse than the syntax-centric approach when that approach produces a true positive while ours produces a false negative (type-3b queries, e.g., which are difficult for attackers to construct). This marginal penalty is more than offset by queries which our approach produces a true positive while the syntax-based approach gives a false negative (type-2b queries, e.g., which are easy for attackers to construct).

We propose to utilize only $k$ tuples from the result set to build the corresponding S-vector. We tested two ways to choose $k$ tuples from a result set.

*Initial-$k$ tuples:* Only the initial $k$ tuples in the result set are used to approximate the entire result set. Statistics computed from these tuples are used to generate the *S-Vector* representation of the query. As soon as the S-Vector is classified as anomalous, we can stop the rest of the pipelined rounds from the database, avoiding extra execution overheads.

*Random–$k$ tuples:* $k$ tuples are chosen at random from the complete result set the S-vector of these $k$ tuples are computed to represent the result set. This approach is expected to produce better accuracy as compared to the inital-$k$ 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 show that our choice of the distance function seems to be *insensitive* to result set ordering, as long as the set is not too small.

### 6.3 Detecting Type 1 and 2a Anomalies, and Masquerade Attacks

This section will show that the data-centric approach works slightly better than the syntax-centric approach for type 1 and type 2a anomalies. The fact that

**Table 3.** Detection Performance – Type 1 Anomalies (Role Masquerade)

| Roles | Algorithm | Syntax-Centric | | | Data-Centric | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **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% |
| | Clustering | 73.3% | 72% | 65.67% | 92% | **92.67%** | **92.33%** | **94%** | **94%** | **92.67%** | **93.33%** |
| 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% |
| | Clustering | **88.5%** | 85.5% | 90.5% | 91.5% | **99%** | **96%** | **98.5%** | **95%** | **100%** | **96%** |
| 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% |
| | Clustering | 78.7% | 73.3% | 78% | **99%** | **100%** | **99.6%** | **99.3%** | **99.3%** | **100%** | **99.3%** |

both approaches work well is to be expected by definition, because both the syntax and the query results are statistically different in type 1 and type 2a anomalies. The syntax-centric scheme in [14] has been shown to perform well in detecting role-based anomalies. Because the results are similar and due to space limitation, we will present only the type-1 anomaly results. Our experiments are also aimed to evaluate the accuracy of both approaches in detecting *role masquerade attacks*. Recall that each query for *GradVote* comes with a user role, and the execution of a typical query in one role by a user with a different role constitutes an anomaly.

*Syntax-Centric Features:* For the sake of completeness, we briefly summarize the syntax-centric data formats of [14]. Three representations are considered: *Crude* (C-quiplet), *Medium* (M-quiplet), and *Fine* (F-quiplet). *C-quiplet* is a *coarse-grained* representation consisting of the SQL-command, counts of projected relations, projected attributes, selected relations, and selected attributes. *M-quiplet* is a *medium-grained* format recording 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* is fine-grained, differing 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 Setup:* The available dataset of queries is labeled by the roles *Staff*, *Faculty*, and *Chair*, in addition to *Framework*, for the common application-generated queries. The query set is randomized and separated into *Train* and *Test* sets of 1000 and 300 queries, respectively. Four query data representations are tested: our *S-Vector* (dimensionality 1638) and the syntax-centric *C-quiplet* (dimen-

**Table 4.** Detection Performance – Type 2b Anomalies (Data Harvesting Attacks)

| **Algorithm** | | **Syntax-Centric** | | | **Data-Centric** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **C quiplet** | **M quiplet** | **F quiplet** | **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)** |
| **Cluster Outlier** | **Detection** | 23.5% | 26.4% | 17.64% | **83.87%** | 12% | 67.7% | 6.4% | 45.1% | 6.4% | 35.4% |
| | **False Positive** | 14.47% | 11.84% | 15.8% | **10.5%** | 3.9% | 6.5% | 3.9% | 5.2% | 2.6% | 6.6% |
| **Attrib Deviation** | **Detection** | 0% | 17.64% | 2.9% | **87%** | **87%** | **87%** | **87%** | **87%** | 12.9% | 64.5% |
| | **False Positive** | 0% | 4.8% | 4.8% | 22.6% | 26% | **15%** | 23.8% | **15.8%** | 23.8% | 20.4% |

sionality 5), *M-quiplet* (dimensionality 73), and *F-quiplet* (dimensionality 1187). Four supervised learning algorithms are tested with each of these feature sets: Naive Bayes, Decision Tree Classifier, Support Vector Machines, and Euclidean $k$-means clustering (see, e.g., [29]).

The results for the binary classifiers for masquerade detection are shown in Table 3 (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. There are two main results. *First*, the performance of the S-Vector based detection using $k$-mean clustering is virtually uniformly better than the syntax-based schemes. In many cases, the detection rates are aproaching 100%. Note also that, the *false positive* rates is the complement of the entries in the table, as there are only two classes. *Second*, the Inital-$k$ and Random-$k$ S-Vector approximations perform very well. This result is important because the Initial-$k$ representation is the most practical one, as alluded to earlier.

It is also noteworthy that the performance of syntax-based schemes is relatively poor using the clustering outlier algorithm. There is one abnormal entry which is the clutering performance of S-V (all) in the "Chair vs. Staff" case, which most likely is due to overfitting.

## 6.4 Detecting Type 2b Anomalies and Data Harvesting Attacks

The focus here is on detecting syntactically similar queries, 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 2b in Table 2.

*Test Setup:* Since type-2b anomalous queries are not available from the real query set, we generate type-2b queries by slightly modifying the normal queries (i.e. queries normally executed by **GradVote** users). Thus, this generated "anomaly set" has approximately the same distribution as the normal queries. Anomalous queries are generated by varying arithmetic and logical operators and constants. As an example, consider the query

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

can be slightly modified to become

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

which yields a vastly different result set.

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 thus not available for training *QStatProfiler*. Hence, supervised learning is not suitable here. We devise two detection techniques based on a single class of *normal* queries: *Cluster-Based Outlier Detection* based on Euclidean-distance clustering, and *Attrib-Deviation* which is a variation of clustering using the $L_\infty$-norm as the distance function.

*Cluster-based Outlier Detection:* The set of queries encountered during the training phase are viewed as points in an $m$-dimensional Euclidean vector space, where $m$ is the dimensionality of the S-vectors. For each user cluster, we select a point in the Euclidean space that is representative of the entire cluster, called the *cluster centroid*, which minimizes the sum of the squared Euclidean distances of the cluster points. For a test vector, the Euclidean distance from the cluster centroid is computed. The query is flagged as an outlier if the vector distance is greater than a specified threshold from any user. In our case, the threshold is chosen to be 3 times the standard deviation.

*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.

A typical performance result with two user roles (*Chair* and *Faculty*) and corresponding anomalous query set is shown in Table 4.

With respect to the performance of the cluster-based outlier detection algorithm, a few points are worth noticing. As expected, the syntax-based schemes show poor performance (since they are essentially 'blind' by design to the Type 2b anomalies). The detection rate for the S-Vector (all) is reasonable (83.87%). However, the Inital-$k$ approximation's accuracy suffers significantly. Upon careful inspection, we find that many of the user queries make extensive use of the SQL *ORDER-BY* clause, which makes the Initial-$k$ statistics unrepresentative of the overall result set statistics. 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 might not be a good choice for this type of anomaly detection. Although anomalies with significant variations in multiple dimensions are easily detected by clustering (as is the case with type-1 and type-2a anomalies), this may not be true with type-2b anomalies. Firstly, Euclidean 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*).

The results for Attrib-Deviation are much better. The syntax based schemes still perform poorly as expected. The data-centric schemes are much better, with detection rates close to 87%, better than the cluster-based schemes. The more important finding is that the attribute-deviation schemes are remarkably resilient to the approximation method. Both Inital-$k$ and Random-$k$ perform as well as the full vector representation; and the Inital-$k$ performs unexpectedly well even with queries generating specific ordering of results.

The resiliency and accuracy of Attrib-Deviation can partially explained as follows. 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 all of them. Thirdly, the selective ordering may not affect all statistical measurements of a single attribute equally (e.g., it may affect *Max*, but not *Median*). It is only when $k$ is very low ($k = 5$) that inital-$k$ performance drops,however Random-$k$ as expected still offers reasonable performace.

We believe that the good performance of the Inital-$k$ 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 (as long as it is not *too* few) initial output tuples. Randomized sampling of query results may not be feasible in general, 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 always 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. It is very difficult for an attacker to craft queries so that multiple statistical measurements are controlled. A theoretical explanation of this intuition is an interesting research problem.

On the minus side, the false positive rates are still too high for the *Attrib-Deviation* schemes. Reducing the false-positive rates while maintaining/increasing the accuracy (true positive rates) is an important research question, which we plan to address in future works.

## 7 Concluding Remarks and Future Work

*Queries:* We construct the S-vectors by expressing the schema of each query result in terms of the attributes of the 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 [30] 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;
```

Here, 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 [31] 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 adequate for certain classes of databases (e.g., US census database), we plan to extend our work to *dynamic* databases. The first challenge is to determine if and when updates shift the boundary between normal and abnormal queries. If the database instance is updated significantly, then the classifiers become obsolete and two things need to be done: (a) detect when a phase shift occurs and re-train, and (b) adopt some form of reenforcement and/or online learing.

For relatively less dynamic databases where updates are less frequent, such as OLAP databases that are heavily used for business intelligence and hence are good targets for insider attacks, it is possible to still apply the data-centric approach, depending on the relative frequency between re-training and data updates. For instance, one can keep a history of legitimate user queries, re-execute them on the new data when the data changes are sufficiently heavy, and use the new result sets to re-train the machine learning model.

Another approach is to separate parts of the schema where data does not change very often and the part that does. Then, the data-centric approach can be applied to the ”projection” of the data space which is static, and the syntax-centric approach can be applied to the dynamic part of the data. This separation can also be done automatically as one can keep track of the statistics of various attributes in the universal table. For example, attributes with high variation over time are more dynamic than others (E.g., Social security numbers, bank accounts of existing customers, dates of births, addresses, and similar fields are mostly static attributes).

*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. 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 and spatial correlations of the query results, 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 is unacceptable from a performance standpoint. The initial-$k$ approximation proposed in Section 6 can help improve performance without sacrificing too much 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 a "shadow query" with the *designated* ORDER BY clause and uses this query result for detection.

Another source of performance improvement might be to design a new statistical model based on both the syntax-based and the data-centric approaches. In cases where we are relatively confident that the syntax-based approach gives a true prositive, we may want to skip the data-centric engine altogether to avoid the database execution. In terms of accuracy, a good combined classifier might perform better too.

Although random-$k$ does not markedly outperform initial-$k$ 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 [32, 33] and queries for data analytics in the area of approximate query answering [34–36].

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. Many open research issues still remain. We aim to develop and present more efficient and practical solutions to these in future work.

## References

1. : Owasp top 10 2007. http://www.owasp.org/index.php/Top_10_2007 (2007)
2. : Owasp-sql injection prevention cheat sheet. http://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet (2008)
3. Schneier, B.: Secrets and Lies: Digital Security in a Networked World. John Wiley and Sons, New York, NY (2000)
4. Bishop, M.: The insider problem revisited. In: Proc. of the 2005 Workshop on New Security Paradigms (NSPW'05). (2005) 75–76
5. CSO Magazine, U.S. Secret Service, CERT, Microsoft: 2007 E-Crime Watch Survey (2007)
6. Cappelli, D.: Preventing insider sabotage: Lessons learned from actual attacks (2005)
7. Brackney, R., Anderson, R.: Understanding the Insider Threat: Proceedings of a March 2004 Workshop. RAND Corp (2004)
8. Schonlau, M., DuMouchel, W., Ju, W., Karr, A., Theus, M., Vardi, Y.: Computer intrusion: Detecting masquerades. Statistical Science **16**(1) (2001) 58–74
9. Chung, C.Y., Gertz, M., Levitt, K.: Demids: a misuse detection system for database systems. In: Integrity and Internal Control Information Systems: Strategic Views on the Need for Control. Kluwer Academic Publishers, Norwell, MA (2000) 159–178
10. Lee, S.Y., Low, W.L., Wong, P.Y.: Learning fingerprints for a database intrusion detection system. In: Proc. of the 7th European Symposium on Research in Computer Security (ESORICS'02). (2002) 264–280
11. Hu, Y., Panda, B.: Identification of malicious transactions in database systems. In: Proc. of the 7th International Database Engineering and Applications Symposium. (2003) 329–335
12. Valeur, F., Mutz, D., Vigna, G.: 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) 123–140
13. Spalka, A., Lehnhardt, J.: A comprehensive approach to anomaly detection in relational databases. In: DBSec. (2005) 207–221
14. Kamra, A., Terzi, E., Bertino, E.: Detecting anomalous access patterns in relational databases. The VLDB Journal **17**(5) (2008) 1063–1077
15. Liu, P.: Architectures for intrusion tolerant database systems. In: Proc. of the 18th Annual Computer Security Applications Conference (ACSAC '02). (2002) 311

16. Wenhui, S., Tan, D.: 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) 249
17. Lee, V.C., Stankovic, J., Son, S.H.: 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) 124
18. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: Proc. of the 10th ACM conference on Computers and Communications Security (CCS'03). (2003) 251–261
19. Srivastava, A., Sural, S., Majumdar, A.K.: Database intrusion detection using weighted sequence mining. Journal of Computers **1**(4) (2006) 8–17
20. Roichman, A., Gudes, E.: Diweda – detecting intrusions in web databases. In: Proc. of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security. (2008) 313–329
21. Fonseca, J., Vieira, M., Madeira, H.: Online detection of malicious data access using dbms auditing. In: Proc. of the 2008 ACM symposium on Applied Computing (SAC'08). (2008) 1013–1020
22. Yao, Q., An, A., Huang, X.: Finding and analyzing database user sessions. In: Proc. of Database Systems for Advanced Applications. (2005) 283–308
23. Ramasubramanian, P., Kannan, A.: Intelligent multi-agent based database hybrid intrusion prevention system. In: Proc. of the 8th East European Conference (ADBIS '04). (2004)
24. Calvanese, D., Giacomo, G.D., Lenzerini, M.: 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) 149–158
25. Maier, D., Ullman, J.D., Vardi, M.Y.: On the foundations of the universal relation model. ACM Trans. on Database Syst. **9**(2) (1984) 283–308
26. Sandhu, R., Ferraiolo, D., Kuhn, R.: The nist model for role based access control. In: Proc. of the 5th ACM Workshop on Role Based Access Control. (2000)
27. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: Proc. of the ACM SIGMOD Conference on Management of Data (SIGMOD '00). (2000) 439–450
28. Kenthapadi, K., Mishra, N., Nissim, K.: Simulatable auditing. In: Proc. of the ACM Symposium on Principles of Database Systems (PODS '05). (2005) 118–127
29. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (October 2007)
30. Stonebraker, M.: Implementation of integrity constraints and views by query modification. In: SIGMOD Conference. (1975) 65–78
31. Buneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: ICDT. (2001) 316–330
32. Olken, F., Rotem, D.: Simple random sampling from relational databases. In: VLDB. (1986) 160–169
33. Chaudhuri, S., Motwani, R., Narasayya, V.R.: On random sampling over joins. In: SIGMOD Conference. (1999) 263–274
34. Haas, P.J., Hellerstein, J.M.: Ripple joins for online aggregation. In: SIGMOD Conference. (1999) 287–298
35. Acharya, S., Gibbons, P.B., Poosala, V., Ramaswamy, S.: Join synopses for approximate query answering. In: SIGMOD Conference. (1999) 275–286
36. Babcock, B., Chaudhuri, S., Das, G.: Dynamic sample selection for approximate query processing. In: SIGMOD Conference. (2003) 539–550