An Analysis of Complexity of Insider Attacks to Databases

GÖKHAN KUL, University of Massachusetts Dartmouth, United States
SHAMBHU UPADHYAYA and ANDREW HUGHES, University at Buffalo, SUNY

Insider attacks are one of the most dangerous threats to an organization. Unfortunately, they are very difficult to foresee, detect, and defend against due to the trust and responsibilities placed on the employees. In this article, we first define the notion of user intent and construct a model for a common scenario that poses a very high risk for sensitive data stored in the organization's database. We show that the complexity of identifying pseudo-intents of a user in this scenario is coNP-Complete, and launching a *harvester* insider attack within the boundaries of the defined threat model takes linear time while a *targeted* threat model is an NP-Complete problem. We also discuss the general defense mechanisms against the modeled threats and show that countering the harvester insider attack takes quadratic time while countering the targeted insider attack can take linear to quadratic time, depending on the strategy chosen. We analyze the adversarial behavior and show that launching an attack with minimum risk is also an NP-Complete problem. Finally, we perform timing experiments with the defense mechanisms on SQL query workloads collected from a national bank to test the feasibility of using these systems in real time.

CCS Concepts: • **Security and privacy** → **Database activity monitoring**; *Intrusion detection systems*; • **Information systems** → *Data management systems*; • **Theory of computation** → Computational complexity and cryptography;

Additional Key Words and Phrases: Complexity analysis, insider threat, query intent, query logs, threat modeling

ACM Reference format:
Gökhan Kul, Shambhu Upadhyaya, and Andrew Hughes. 2020. An Analysis of Complexity of Insider Attacks to Databases. *ACM Trans. Manage. Inf. Syst.* 12, 1, Article 4 (December 2020), 18 pages.
https://doi.org/10.1145/3391231

1 INTRODUCTION

Cyber attacks have become an increasingly prevalent problem for organizations. Although considerable budgets are allocated for the information security departments to detect and respond to cyber attacks immediately and efficiently, the threat persists [12]. One of the most critical types of cyber attack is the *insider attack*, which occurs when employees misuse their legitimate access

This material is based in part upon work supported by the National Science Foundation under award number CNS - 1409551. Usual disclaimers apply.
Authors' addresses: G. Kul, University of Massachusetts Dartmouth, 285 Old Westport Road, Dartmouth, MA, 02747; email: gkul@umassd.edu; S. Upadhyaya and A. Hughes, University at Buffalo, SUNY, 338 Davis Hall, Buffalo, NY, 14260; emails: {shambhu, ahughes6}@buffalo.edu.

2158-656X/2020/12-ART4 $15.00
https://doi.org/10.1145/3391231

rights or gain unauthorized access to a resource, such as file systems and servers [7]. Unauthorized account openings and money transfers, identity theft, and credit fraud are just a few other well-known examples of insider attack.

The "2018 Global State of Information Security Survey" [38] states that 30% of the security incidents originated from current employees while 26% of the incidents originated from former employees. Same study points out that it takes 50 days on average to resolve a malicious insider attack *if* it is detected. "The Deloitte 2019 Future of Cyber Survey" [31] reports that even the actions of well-meaning employees are the second-most-concerning cyber threat to organizations with 32% among the participants. Another study [37] surveyed a range of cyber attack types and determined that insider attack is the most costly attack type to an organization. Financial organizations took the hardest hit from insider attacks, costing the highest annualized worth compared to other sectors.

The difficulties of dealing with insider attacks are threefold. First, we are dealing with trusted employees who have inside knowledge about the organization, security infrastructure, and information flow. Second, creating fine-grained and restrictive access policies for shared resources restrict a legitimate actor's ability to adapt to new or unexpected tasks, while permissive policies allow room for exploitation. Last, a new legitimate activity from a legitimate actor can be perceived as an anomaly although the intent is benign. For example, a teller in Florida who withdraws a large sum for a client with an address from Montana may be acting legitimately (e.g., if the client is on vacation and needs to pay an emergency-room bill) or may be committing fraud.

Current insider threat detection mechanisms usually depend on detecting anomalies in a user's behavior, focusing on a specific type of resource or a combination of resources [40]. These resources can be shell commands [36], file accesses [34], and SQL queries [20, 26, 30, 33]. Another branch focuses on psychological factors instead of resource usage [6].

In this article, we analyze insider threats against relational databases by focusing on the complexity aspects of most common threat models researched in the literature, along with the state-of-the-art defense mechanisms to prevent them. Although these methods prove to be effective in the area they are developed for, the literature survey shows the lack of accurate *modeling* of insider threats in this domain, an analysis of computational complexity for user intent modeling, and a generalized strategy for dealing with such attacks. We bridge this gap by constructing the notion of user intent model and creating a generalized threat model for database systems. We evaluate the time complexity of launching an attack using the modeled threat and discuss the defense mechanisms to counter these while considering the time complexity. Our experimental analysis focuses on time complexity and time feasibility of using these mechanisms rather than their accuracy and precision. There are several database workload-based anomaly detection systems that are used for attack detection in the literature [13, 22, 26, 41, 42, 45, 46] that provide in-depth discussions of accuracy and precision. Many of the systems that use SQL queries for attack detection have common operations such as parsing and clustering. Although the literature has comprehensive experiments on attack detection, the feasibility of these defense systems in terms of timing performance and complexity has not been discussed. Since this article is specifically interested in the timing property of the database log classifier, we test if parsing and clustering operations perform reasonably fast to be practical for use in a corporate setting. Additionally, most studies use synthetic datasets or real query workloads that do not reflect real-world corporate settings. For these reasons, we collaborated with a national bank headquartered in the United States to collect SQL query logs from their database servers.

Concretely, the contributions of this article are:

(1) Defining the notion of intent model for database users and a systematization of knowledge in this area,

(2) Building a threat model for insider attacks in the context of relational databases,
(3) Analyzing the timing aspect of counter actions against the modeled threat, and
(4) Providing a computational complexity analysis for both the threat and defense mechanisms.

This article is organized as follows: We start by providing background on database systems and a broad state of database security research in Section 2 and mathematically formulating the intent model in Section 3. We then describe the threat model in Section 4. We present the complexity analysis for harvester and targeted attacks from the perspectives of both threat and defense models in Section 5. We perform timing experiments for parsing and clustering of queries—main functions used in detection systems—in Section 6. We discuss other factors that can be utilized in both threat and defense models in Section 7. We conclude, and briefly present our future work, in Section 8.

2 DATABASES AND SQL QUERIES

A *relational database* is a set of relations (i.e., *tables*), and a *relation* is a bag or set of tuples (i.e., *rows*) where a tuple is a structured data item. The structure is defined with attributes (i.e., *columns*) and the types of the attributes. The difference between bag semantics and set semantics is that bag semantics allow duplicate records whereas the other enforce uniqueness.

Relational Database Management Systems (RDMS) can combine multiple resources efficiently and ensure to provide accurate results. The most important principle in these systems is that a query simply must not return a wrong result.

Structured Query Language (SQL) is a declarative language that is designed for managing, manipulating, and retrieving from relational databases. Other than schema manipulation and data access control operations, SQL queries mainly perform four different operations: (1) insert, (2) update, (3) delete, and (4) select. The basic structure of these operations is as follows:

```
(1) INSERT INTO table (column1, column2, ...)
                VALUES (value1, value2, ...);
(2) UPDATE table
    SET column1 = integer|decimal|string|...
    WHERE column2 = integer|decimal|string|...
(3) DELETE FROM table
    WHERE column1 = integer|decimal|string|...
(4) SELECT [aggregation] column1, column2, ...
    FROM table1, table2
    [WHERE table1.column1 = table2.column3]
    [ORDER BY column1]
    [GROUP BY column1]
    [LIMIT integer]
```

where the brackets show optional query items, and the upper-case items are the *keywords* that identify the operations to be performed. As can be implied from these basic structures, queries that perform similar tasks often have analogous structures or at least share some attributes.

SQL query statements are constructed from *clauses*. Every line of the query structures given above constitutes a clause. As an example, let us take the following query:

```
(1) SELECT u.username, u.yearenrolled
(2) FROM user u, accounts a
(3) WHERE u.id = a.userid AND a.balance > 1000
```

```
(4) GROUP BY u.yearenrolled
(5) ORDER BY u.yearenrolled
```

Line 1 consists of the `SELECT` keyword, and the *projection* items. Line 2 has the `FROM` clause that lists the tables the query is going to use. Line 3 is named the `WHERE` clause. Where clause contains *selection* and *join* expressions. `u.id = a.userid` expression is a join expression, and `a.balance > 1000` is a selection expression. Lines 4 and 5 include the *group-by* and *order-by* items, respectively.

3 INTENT MODEL

The SQL queries that a user issues on a database could model the normal usage behavior [13]. Also, queries that are similar in nature imply that they might be issued to perform similar duties [26]. However, understanding the intent of the query is regarded as being as hard as constructing a new query, and it gets even more complex when the query is complicated [17]. For this reason, capturing the intent of a query is often ambiguous [2], and there can be various feature extraction methods. In the literature, SQL feature extraction to capture the intent of a query has been studied for different purposes such as performance optimization [4], workload analysis [2, 32], query recommendation [10, 24, 48], and security purposes [20]. For more specialized and complicated information needs, the construction of the query may intrinsically get complicated, which often makes it troublesome to understand the resulting query for human readers. It gets harder to compare the similarities of the queries in terms of accuracy when the complexity of the question increases, even though they are created to accomplish the same task [2]. For this reason, in our previous work [27], we analyzed clustering quality of workloads that consist of queries with the similar intents and proposed a *regularization* method to increase clustering quality.

When we take a closer look at these feature extraction methods, they essentially extract *projections*, *selections*, *joins*, *group-by*, and *order-by* items from the SQL query, or a subset of them, to be used in query similarity comparison. The similarity metric varies based on the aim of the extraction method: Sometimes distinct queries produce the same set of features, and sometimes queries that aim to perform the same task can produce different sets of features [27]. Of course, the results of queries also depend on the data stored in the database. For example, the queries

```
(1) SELECT * FROM user
    WHERE username LIKE "A%"
(2) SELECT * FROM user
```

will produce exactly the same result if all the `username` values start with "A" in the `user(firstName string, lastName string, username string)` table. Hence, a query can be perceived with varying *interpretations*, which makes it impossible to extract the definitive *intent* of the query writer from a given query, but it is still possible to sense a fuzzy notion of the aim.

We equate this fuzzy notion with the *pseudo-intent* concept in Formal Context Analysis (FCA). A *formal context* is a triple $\mathbb{K} = (\mathbb{G}, \mathbb{M}, \mathbb{I})$, where $\mathbb{G}$ is a set of *objects*, $\mathbb{M}$ is a set of *attributes*, and $\mathbb{I}$ is a *relation* that associates each object g with the attributes satisfied by g. To express that an object g is in relation $\mathbb{I}$ with an attribute m, we write $g\mathbb{I}m$ [5].

Identifying if a subset of attributes is a pseudo-intent is shown to be coNP-Complete [5]. It is also important to note that not all sets of attributes in a context represent a pseudo-intent; counting the number of pseudo-intents is proven to be #P-hard, while finding the number of sets that are not pseudo-intents is shown to be #P-Complete [14].

Thus, to understand the pseudo-intent, we extract the relevant features from SQL queries, and following the definition provided by FCA, we define the attributes as the SQL query features.

These attributes are essentially the resources consumed by the SQL query. We will refer to these resources when we say *query intent*—(*pseudo-intent in FCA domain*), from now on. In our previous work [27], we show that the query extraction method applied affects the quality of clustering, hence the interpretation of the query intent.

Definition 1 (Query Intent). Under the assumption that resources consumed by a user to perform a task reflect the user's intent, we define intent as a finite bag of resources denoted by $\phi = \{r_1, r_2, \ldots, r_{|\phi|}\}$.

This definition directly conforms with the feature extraction methods described above where r_i is the extracted query feature. For instance, Makiyama method [32] would extract the feature set of query (1) above as $\phi = \{$`SELECT_firstName, SELECT_lastName, FROM_user, WHERE_username`$\}$. Further information is available in the literature [27] on the analysis of how the query features affect clustering quality and how they can be engineered to better match queries with similar pseudo-intents.

Definition 2 (User Activity). User activity A is represented by a user $u \in U$, where U is the set of all users, for the time period T that starts from t_0 and goes on for Δt, and the set of intents ϕ performed by u within T. Formally,

$$A_u^T = \left(a_u^{t_0}(\phi), a_u^{t_1}(\phi), \ldots, a_u^{t_n}(\phi)\right), \tag{1}$$

where $a_u^{t_i}$ represents a timestamp of an activity performed by user u.

The users usually create similar workloads on the database to perform their daily tasks [26]. These workloads can be utilized to create a chain of tasks for each user.

4 THREAT MODEL

Insider threat is defined as "malevolent (or possibly inadvertent) actions by an already trusted person with access to sensitive information and information systems [3]." It is considered as one of the most prevalent security problems for intelligence community, military, and finance organizations. Unlike an attacker who is trying to access information from outside, an insider usually has knowledge about the database and procedures within the organization. Using this knowledge, they can exploit the privileges and trust placed on them and access information that they are not supposed to see. Insider attacks can occur in many forms:

Privilege abuse attacks. The adversary retrieves information with their existing access permissions without needing to know to perform their duties [28]. In this attack type, the adversary gleans or tampers information for a malicious purpose on the database that they have legitimate access rights.

Masquerade attacks. The adversary gathers the credentials of a legitimate user to access the system. Using these credentials, they assume the identity of the victim, and access the database [28]. In this attack type, a user assumes the roles of another user. The malicious activity of the adversary on the database is logged with the credentials of the victim.

Privilege escalation attacks. The adversary, either by exploiting a vulnerability of the system or by social engineering, escalates their permissions to access more sensitive data that would be unavailable to them otherwise [28]. The adversary uses their own or someone else's credentials to elevate their access rights.

It is also worth mentioning that an **external adversary** can penetrate the defense mechanisms such as firewalls in place via various techniques and can gain credentials of a legitimate user. Any attack techniques and defense mechanisms up until that point are out of the scope of this article.

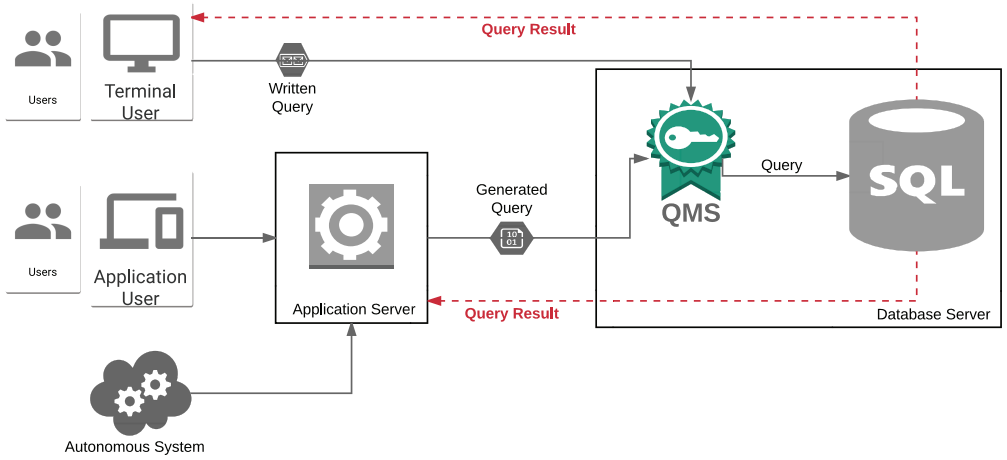


Fig. 1. An architecture of query monitoring.

However, after that point, depending on the actions of the adversary, the attack can turn into one of the three insider attack types mentioned above. Once an external adversary gains access to the credentials of a legitimate user, they can glean or even manipulate sensitive information if the credentials have access rights. This poses a unique threat, since the observed queries by that user account do not actually belong to the account owner, which corresponds to a masquerade attack. A capable external attacker can stage other attacks such as a privilege escalation attack at this point.

Stallings [44] classifies cyber-attacks as *passive* and *active* and defines them as follows: "*A passive attack attempts to learn or make use of information from the system but does not affect system resources. An active attack attempts to alter system resources or affect their operation.*"

In this article, we focus on the data leakage threat posed by insiders. Most research in the literature in this domain considered *passive attacks* [18, 20, 22, 23, 26, 30, 41, 42, 46]. The common point in all of the threat models is that the attackers query the database to extract sensitive information but they do not tamper with the data. They may access the database system through a client-side application or through direct interaction with the database server while still having the queries observed by the query monitor. Also, the solution methodology follows the same pattern: The monitoring mechanism depends on detecting anomalies in the user's database usage. The suspicious activity detection usually consists of the following steps: (1) extract relevant features that reflect the user behavior, (2) cluster or classify similar actions, and (3) find outlier actions, which presents an effective approach [20, 29]. Since this methodology proves to be a common approach, and its accuracy and effectiveness have been tested in many studies [13, 22, 26, 41, 42, 45, 46], we focus on the timing aspect of parsing of the query for feature extraction and clustering with two common clustering methods, k-means and hierarchical, in Section 6.

See Figure 1 for an architecture that uses a query intent model for monitoring database activity and to flag potential insider attacks. The application users interact with the database using a web application on their computer (client-side). This application generates queries and issues them to the database, which is contained in a database server (server-side). The terminal users interact with the database via a terminal interface on their computers, which directly connects them to the database server. The query monitor system (QMS) just observes the queries that are issued to the database, and it does not block or change the queries. Any query that is issued to the database is captured by QMS, processed, logged, and then sent to the database, respectively. Although

the query monitor does not block any queries, it detects suspicious activity and reports them to the security personnel. Consequently, QMS just observes the interaction between users and the database, but never modifies the query results or database records.

In the rest of the article, we consider two passive attack models: harvester (*a.k.a. aggregate*) and targeted (*a.k.a. individual*) attackers [35]. The harvester attacker is an adversary who manages to replicate one part of the database for exploratory analysis by querying the database. The targeted attacker is, however, an adversary who accesses certain information without needing to know for legitimate purposes, but chooses the attack parameters carefully to avoid being detected.

We do not address *active attackers*, who tamper with the data or query results, because we only aim to prevent identity theft and information leakage type threats posed by insiders by stealing information. To address data tampering by an insider, the detection system can be supported with integrity verification techniques [21] and authenticated data structures [49]. We also do not address *snapshot attacks* in which the adversary copies the database server or the database instance completely. Such attacks cannot be caught by the query monitor, since it does not have any control on the server instance. The cloned server instance, if the attack is successful, would be available for use of the adversary. In that case, the adversary would not need to go through the query access monitor to query the database. This is an offline attack in the sense that the attacker steals the database and works offline for as long as they wish to extract data out of it.

5 COMPLEXITY ANALYSIS

Consider a query Q_1 issued on a database D. To find the tuple t that the user requests with the selection clause $\sigma_{s=c}(R) = t$ where s is the attribute name in table R and c is the constant the query is searching for, the database system scans through all the values on $R.s$, which is performed in $O(|R|)$, namely, in linear time. Of course, this can be made faster; for example if there is an index built on the column previously, the evaluation of Q_1 can be improved up to $O(log|R|)$ time. However, as the intent gets more complex, the evaluation takes more time. It is known that query evaluation problem is NP–Complete for conjunctive queries [9], and the space complexity of query evaluation problem for relational algebra, in general, is PSPACE–Complete [25].

Next, we analyze the modeled threat and derive several computational complexity results pertinent to the model.

5.1 Analysis of Harvester Attacks

Harvester attacks focus on collecting a variety of information from the database, which result in search behaviors that show high levels of *diversity* and *broadness* according to the state-of-the-art solution presented by Wang et al. [47]. Diversity is measured by the query and parameter similarity within a database session while broadness is measured by the variety of the return results from the queries within a database session. In the rest of this section, we discuss the threat and defense models based on the approach presented earlier.

5.1.1 The Harvester Threat Model. We assume that the adversaries within the organization have the domain knowledge about the database schema or have access to software tools that run on the database without requiring the user to have familiarity to the underlying database, but they do not have any insight into the data content stored in the database. When the attackers are not looking for specific information, they can issue exploratory queries on the system. This can be in two forms: (1) none or few filtering conditions or (2) a lot of queries with filtering conditions. Since this attack type does not have a specific target information, the adversary has only one goal: extracting as much information as possible. In the most basic form, the adversary would issue wildcard queries

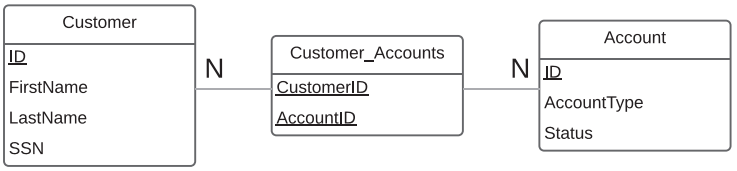


Fig. 2. MY_BANK database diagram.

for each table, and they can retrieve the entire database by issuing n queries where n is the number of relations in the database.

Consider a database MY_BANK with relations CUSTOMER, which includes details about the customers of a bank; ACCOUNT, which contains all the accounts the bank is handling with many-to-many relationship; and CUSTOMER_ACCOUNTS, which has the information of which accounts belong to which customers as shown in Figure 2. An insider who has information about MY_BANK schema can access all the data in the database with only three queries: `SELECT * FROM customer`, `SELECT * FROM account`, and `SELECT * FROM customer_accounts`.

It is also possible for the adversaries to utilize filtering conditions, which would result in increasing the number of queries to avoid using wildcard queries to protect themselves from detection. However, the query monitor, in the end, would log the resources accessed while evaluating the query and would eventually end up logging the same resources in both cases. Hence, although there can be a lot of ways to harvest all the data, the complexity would still be $O(|R|)$.

5.1.2 The Harvester Defense Model. Analyzing queries mostly relies on the structure of queries [20], since access to the data in the DBMS may not always be possible for auditing systems or people responsible for investigating attacks. Systems that utilize query correlation usually exploit the resources the queries want to access as mentioned in Section 3, and, hence, the features in the intent set can be used to measure the diversity. Data-centric query comparison [33], however, requires access to the data and can be time-consuming, since it involves query evaluation. After the query evaluation phase, the search results of the queries should be compared to measure the broadness.

For instance, Makiyama et al. [32] approach query log analysis with a motivation of analyzing the workload on the system. They extract the query terms in selection, joins, projection, from, group-by, and order-by items separately and record their appearance frequency for each query in the dataset. They create a feature vector using the frequency of these terms, which they use to calculate the pairwise similarity of queries with cosine distance function. The feature extraction method presented in this work can be used to measure diversity [20]. As we indicated before, for the logging mechanism, there is no difference between using wildcard queries and using the resource names directly in the query. For example:

```
Q1: SELECT * FROM customer
Q2: SELECT ID, FirstName, LastName, SSN
    FROM customer
```

When the query parser receives the query $Q1$, it processes the * as ID, FirstName, LastName, and SSN. Hence, the query similarity function used would consider $Q1$ and $Q2$ as the same and $\phi_{Q1} = \phi_{Q2} = \{ID, FirstName, LastName, SSN\}$.

The complexity of feature extraction from the query can be considered $O(1)$. To create the intent for all queries ϕ_{ALL} in a user's session, we should process all the queries in that session. Hence, the time complexity to create the intent set is $O(n)$, where n is the number of queries issued. To

compute the diversity, we compare the resources in ϕ_{ALL} with all the possible resources (S) in the database. The time complexity of this operation is $O(|\phi_{ALL}| \cdot |S|)$.

5.2 Analysis of Targeted Attacks

Targeted attacks focus on specific information in the database and investigate for sensitive information that the attacker is interested in. The activities can be similar to the common and legitimate activities of the user. However, the attacker acts without needing to know for any legitimate purposes [20, 26, 45].

5.2.1 The Targeted Threat Model. Any user, including an insider, should consider not only the data retrieved including the needed data, but also should consider retrieving only the most important results when issuing the query to the database. Knowing that issuing too many queries on a database could raise suspicions, a crafty intruder should be precise about the information that they need to achieve their goal. Thus, the retrieved rows should provide the maximum coverage and minimum redundancy. Still, the question remains if the resources that the insider wants to access are in the database and if they can be retrieved with accessing a limited number of resources. The motivation for this limitation can vary; the needed time to answer a query can dramatically increase as the complexity of the query increases [32], the access policy can prevent certain information from being used together [1], or accessing too much information can raise an alert [16]. Hence, the insider should be precise while preparing each expression and avoid redundancies.

Considering that the expressions included in the query reflect the intent of the attacker, we can use our intent model—$\mathbb{K}$—presented in Section 3 as a base while formulating the problem:

Question 1 (Intent Set Problem). *Every time a user issues a query, given a limitation of maximum number of resources that can be accessed, does there exist a query construction that will return the information that the user is looking for on the system?*

Claim 1. *The construction of the SQL query with the user's intent is NP–Complete.*

Proof. Let V_1 be the feature set that includes all possible columns and constants. Let V_2 be the user intent set of an attacker. Let k be the maximum number of resources that can be accessed. We can construct a graph $G = (V, E)$ where $V = V_1 \cup V_2$. Note that $V_1 \cap V_2 = \emptyset$.

For any $u \in V_1$ and $v \in V_2$, we include the edge $(u, v) \in E$ if the expression v includes the item u. An intent set is a set $I \subseteq V_1$ such that every node in V_2 is a neighbor of at least one element of I; namely, for each $v \in V_2$, there exists $u \in I$ where $(u, v) \in E$.

The problem of *Intent Set* is to determine for a tuple (V_1, V_2, E, k) if the graph $G = (V_1 \cup V_2, E)$ contains an intent set of size at most k. *Intent Set* is in NP. Given a tuple (V_1, V_2, E, k) and a candidate intent set I, the verification of it being an intent set can be performed by checking if $I \subseteq V_1$, each element in V_2 has a neighbor I. This operation can be performed in polynomial time.

We can show Intent Set is **NP–Complete** by reducing Set Cover, a known NP–Complete problem, to Intent Set. Let (U, S, k) be an instance of Set Cover, where U is a set of n elements, S is a collection of m subsets of U, and k is some value $1 \leq k \leq m$. Set Cover asks if there exists a group of k or fewer sets from S such that their union equals U. Let $V_1' = S$, $V_2' = U$, and (s, u) is in E' if $u \in s$. Then (V_1', V_2', E', k) is an instance of Intent Set. If $G' = (V_1' \cup V_2', E')$ has an intent set of size k or less, then it directly corresponds to a covering of U of size k or less. If G' does not have an intent set of size k or less, then U cannot have a covering of size k or less. Therefore, Set Cover reduces to Intent Set in polynomial time.

As Intent Set belongs to NP and Set Cover is NP–Complete [19] and reduces in polynomial time to Intent Set, Intent Set is NP–Complete. □

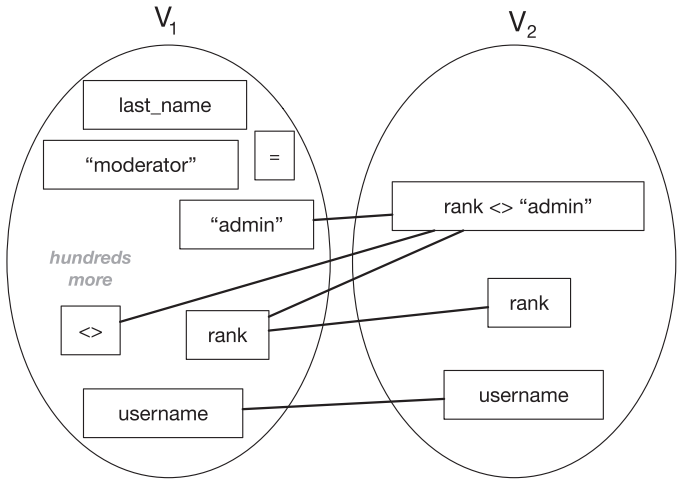


Fig. 3. Intent Set for `SELECT username, rank FROM user WHERE rank <> "admin"`.

Hence, given the relation between the user's intent and the resources required for each intent, the problem to determine if a set of k resources provides access to the desired intent is NP-Complete. The general intuition this proof conveys is that, without having knowledge about the contents of the database, and allowing access to only k number of resources, a query should be constructed very carefully by an attacker to avoid risk of detection. However, as the user intent grows, the restriction k imposes will make it more challenging, possibly intractable, to construct a query that provides access to the target data.

Example. Non-admin users

For illustration purposes, let us consider Ashley, an adversary. She decides to gather sensitive information of some users who are not administrators for a masquerade attack. To find such data, Ashley needs to issue a query that filters the rank of the users and brings sensitive information about them that she can use. Let us assume that the current credentials that Ashley uses limit her to access at most four resources.

There are many factors that she has to consider: "*What credentials can be useful?*," "What resources can be used?," "Does the data she is looking for exist in the database?," and so on. She can come up with a query such as `SELECT username, rank FROM user WHERE rank <> "admin"`.

When we apply this method to the query `SELECT username, rank FROM user WHERE rank <> "admin"` as shown in Figure 3, the intent set we need to use is

$$I = \{username, rank, <>, \text{“}admin\text{”}\}$$

Consequently, she would be using four resources, and the query given would provide usernames of non-admin users. However, if the credentials she uses do not have access rights to at least four resources, she will not be able to issue this query without getting caught.

Approximation. According to Claim 1, Intent Set is NP–Complete. However, there is a greedy algorithm (Algorithm 1 below) that gets an approximation ratio of $ln\ n$, with loosened constraints. Consider that the attacker has access to a benign query set (i.e., query log). In that case, the attacker can pick the query that covers the most points in V_2. The attacker throws out all the points that the picked query covered, and repeats.

Claim 2. *If the optimal SQL query construction uses k sets, the greedy algorithm finds a solution with at most $k * lnn$ sets.*

ALGORITHM 1: Greedy Intent Set Algorithm

```
1: procedure GREEDY-INTENT-SET(V1, V2, k)
2:     U ← V2
3:     C ← emptyset
4:     while r ≠ 0 AND k * lnn ≥| C | do
5:         Select an S ∈ V1 that maximizes | S ∩ U |
6:         U ← U − S
7:         C ← C ∪ {S}
8:     end while
9:     return C
10: end procedure
```

Proof. The optimal query construction uses k sets. Consequently, there are some sets that cover at least $1/k$ of all the points in V_2. If we choose a query that covers the most points in V_2, it will cover $1/k$ points or more. This leaves $n * (1 - 1/k)$ points in the intent set. When we repeat the same operation m times, it leaves $n * (1 - 1/k)^m$ points. As a result, in at most $m = k * lnn$ rounds, our operation will complete. If $P \neq NP$, this is the best approximation ratio we can expect [15]. □

The implication of this algorithm is that, the attackers who have read access to query logs can use this information to construct their attack strategy. Following this approximation algorithm, the attackers may stop at a point where they are satisfied with the information they accessed without needing to construct the optimal SQL query.

5.2.2 The Targeted Defense Model. The distinctive traits of targeted attacks are that the adversaries access information without needing to know and likely deviate from their normal behavior. There are two approaches to mitigate these attacks: (1) misuse detection and (2) anomaly detection. Misuse detection focuses on detecting predetermined specific malicious activities. The defense mechanism creates rule sets on what kind of behavior a user should not perform, which requires fine-grained security analysis on the system and identifying what information each user should not have access to. Anomaly detection focuses on deviations from normal patterns.

Misuse detection mechanism is a rule-based system, which checks every activity of a user to determine if it matches with the forbidden behavior defined in the rule set. This check is usually straight-forward and takes linear time.

Anomaly detection mechanism, however, forms *user profiles* to formulate a normal behavior pattern for each user. We take every query issued by all users and extract the resources consumed to use them to create the user profiles. After that, we can approach this problem as a clustering problem. The information collected is labeled with a clustering algorithm like k-means or hierarchical clustering while keeping the resource-user association. A user profile can be created from the distribution of labels to each cluster. Utilizing this information, the anomaly detection mechanism observes each query issued by each user and catches the anomaly if a user's distribution starts to shift from the profiled distribution [20, 45].

Clustering-based anomaly detection systems require a one-time clustering operation to create profiles. This operation can be repeated to update the user profiles as time progresses. The time complexity of the clustering operation depends on the selected technique; if it is a technique that requires a pairwise distance matrix, the operation has quadratic complexity, and if it is a heuristic-based technique like k-means, the operation has linear complexity [8].

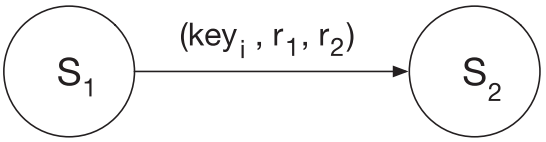


Fig. 4. Key risk representation.

Hence, heuristic-based k-means clustering requires less computation time, which makes it a better choice when there is a need to update the training data frequently. However, accurately determining the k value, namely, the number of clusters, requires thorough analysis of the organization, user behavior, and attack scenarios. However, hierarchical clustering requires more computation time, especially as the data size grows, but adjusting the cluster labels is easier after the clustering is completed.

In our timing experiments in Section 6, we measure the time required for parsing queries and performing hierarchical clustering and k-means clustering to test if these methods are practically feasible in a corporate setting with varying sizes of data. Our observations shown in Figure 7 are consistent with the complexity assessment.

5.3 Analysis of Adversarial Behavior

The adversaries launch an attack when they are convinced that the time and effort spent and the risk taken to reach the information are worth the value of the information [43]. We assume that the adversary, being an insider, knows that there are security measures in place against insider attacks. They can research on the defense strategies and inquire about the system beforehand to find out the weaknesses to reduce the risk of exposure. However, while trying to reduce the risk, the adversaries still need to gain access to some *key information* to reach their target.

In this section, we describe a model called *key risk graph* that shows different paths that the adversaries can take to reach their target. We can naively describe every step in this path as follows, and as represented in Figure 4:

- State representing the adversary's current status, S_1;
- State representing the adversary's desired status, S_2;
- Key information needed to go from S_1 to S_2, key_i;
- Risk associated with the move if the adversary has key_i, r_1;
- Risk associated with the move if the adversary does not have key_i, r_2.

Definition 3 (Key Risk Graph). Key Risk Graph KG is a directed graph that shows the adversary's informational states, key information needed to change the current state to another, and risks associated with going from one state to another.

$KG = (V, E; \beta, V_0, V_S, \pi, \delta)$ where V represents the set of all possible states, and E represents the set of navigations from one state to the other. β is the set of key information, V_0 is the starting state, V_S is all the other states that can be traversed only if the risk associated with one of the incoming edges is taken by the adversary, $\pi : V \rightarrow \beta$ is a function that assigns keys to the states, and last, $\delta : E \rightarrow \beta \times \mathbb{N} \times \mathbb{N}$ is a function that assigns risks to the edges where $\mathbb{N}$ represents all natural numbers.

Given a Key Risk Graph, the problem of finding an attack with the minimum risk is **NP-Complete**, since there is a direct mapping from KEYSEQ problem [11], where the authors provide a proof with a reduction from 3–SAT to KEYSEQ.

To further illustrate how the Key Risk Graph can be utilized in practice, consider a plausible real-life scenario as below.

Table 1. Distribution of Query Types in Bank Dataset

Type	Queries
Select	1,349,861
Union	1,306
Insert	1,173,140
Update	288,098
Delete	6,314
TOTAL	2,818,719

Table 2. Summary of Query Log Size by Absolute Time

Time	Total Time	SELECTs	Skeletons
12:34 PM	1.5 hours	250,740	756
2:24 PM	3.3 hours	512,981	946
4:06 PM	5 hours	789,050	1,057
6:27 PM	7.3 hours	972,511	1,102
9:23 PM	10.3 hours	1,079,427	1,130
12:43 AM	13.6 hours	1,176,582	1,256
04:04 AM	17 hours	1,270,984	1,576
6:28 AM	19.4 hours	1,349,861	1,614

Example. Identity Theft and Credit Fraud

Actors: Alice (Adversary), Bob (Victim), Celia (Victim), Dan (Collaborator)

Bob, the customer, goes to a bank to payout his auto-loan. While explaining why he is there, he mentions that he and his wife sold a house that they just inherited, and they are paying their debts using that money. Alice, the banker, figures the rest of the money must be in his wife's account. After completing the transaction, Bob leaves the bank. Alice decides that if she can find out the SSN number and birth date of Bob's wife, she can use that information to apply for a loan collaborating with her contact, Dan, in the credit department. They pay off the loan using the money in the account and take the loaned money for themselves. There are two ways that she can access that information:

- *Querying the home address information Bob provided, she finds out there are two people living at the same address: Bob and Celia.*
- *Querying the auto-loan information, she finds out Bob's co-signer is Celia, who carries the same last name.*

Alice determines Celia must be his wife, so she looks up Celia's account information, including her birth date and SSN, and passes the information to Dan.

In the example given above, Alice tackles two challenges to reach the key information she needs: Finding out who Bob's wife is, and using the information she found, she reaches her target. Assuming that she knows how the defense mechanism described in this article works, she evaluates both strategies. The first strategy involves looking up an address that is usually not a common practice, hence, it is associated with a higher risk (r_2) of triggering the security system. The second strategy, checking the auto-loan information (key_i), however, is just a very common procedure while paying off a loan, hence, she finds it less risky (r_1).

6 EXPERIMENTS

In the previous sections, we focused on the time complexity of database attacks and defense mechanisms. In this section, we would like to test if parsing and clustering, which are essential operations in SQL log classifiers used in database security, can in fact perform reasonably fast to be practical for use in a real-world corporate setting. To collect data from a real-world corporate setting, we collaborated with a national bank headquartered in the United States to collect SQL query logs from their database server.

The dataset includes SQL query logs that capture all query activity on the majority of databases at a major U.S. bank over a period of approximately 19 hours. Logs are best-effort anonymized by replacing all constants with hash values generated by SHA-256. We are only able to share the

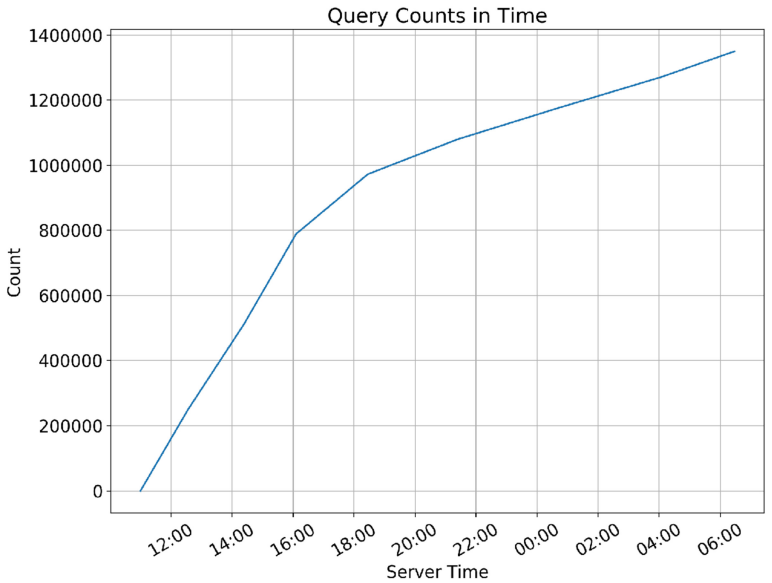


Fig. 5. Server query workload.

summary of the dataset in Table 1 due to the sensitivity of the information and querying data. There are 2,818,719 parsable queries in our dataset. Since we only inspect the data access patterns, we base our analysis on the 1.35M syntactically valid SELECT queries.

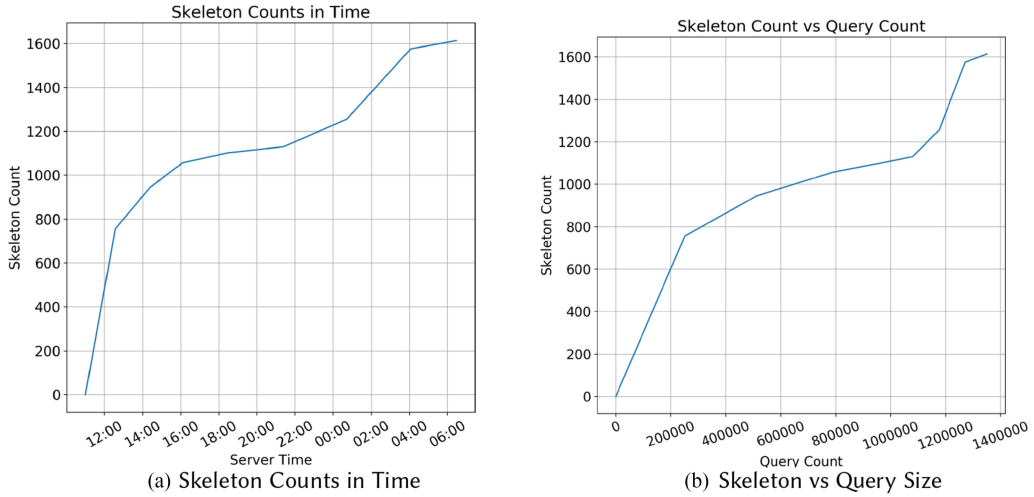
To test scalability, we vary the number of queries processed by truncating the log after varying lengths; in the first 1.5 hours, the log contains approximately 250,000 SELECT queries with 756 skeletons, just over 500,000 queries after 3.3 hours, and so forth. Full statistics for each truncated log are shown in Table 2. The query workload is shown in Figure 5.

A query with its constant values replaced by a placeholder grammar term is called a *query skeleton*. Our first observation is that there are only 1,614 different query skeletons. This means that the number of distinct structures that we need to cluster is quite small. Furthermore, this number grows sub-linearly over time. As shown in Figure 6(a), nearly half of all the skeletons arise in the first 1.5 hours of the trace. As the query log size grows, these queries are converted to skeletons for clustering.

The time measurements were taken on a commodity computer with a 3.2 GHz Intel Core i7 processor and 16 GB RAM memory running Ubuntu 14.02. The parsing of the queries was performed with a modified version of JSQLParser.[1] The modified version[2] was developed at the University at Buffalo. To be able to handle varying SQL notations, we encountered that the original parser was not able to parse at the time of the experiments. We used the R implementation of hierarchical and k-means clustering. The remaining components were implemented in Java using the single-threaded JDK 1.8.0.[3] Reported running times are the average of 10 trials for each phase.

The running time of query parsing grows sub-linearly with the log size, as can be seen in Figure 7. We attribute this to the dominant costs being resource allocation associated with defining and indexing new query skeletons and potentially JIT compilation overheads. The running time of hierarchical clustering grows super-linearly with the number of query skeletons. This is typical

[1] https://github.com/JSQLParser/JSqlParser.
[2] https://github.com/UBOdin/jsqlparser.
[3] https://github.com/PADLab/PocketPattern.



(a) Skeleton Counts in Time (b) Skeleton vs Query Size

Fig. 6. Skeleton conversion.

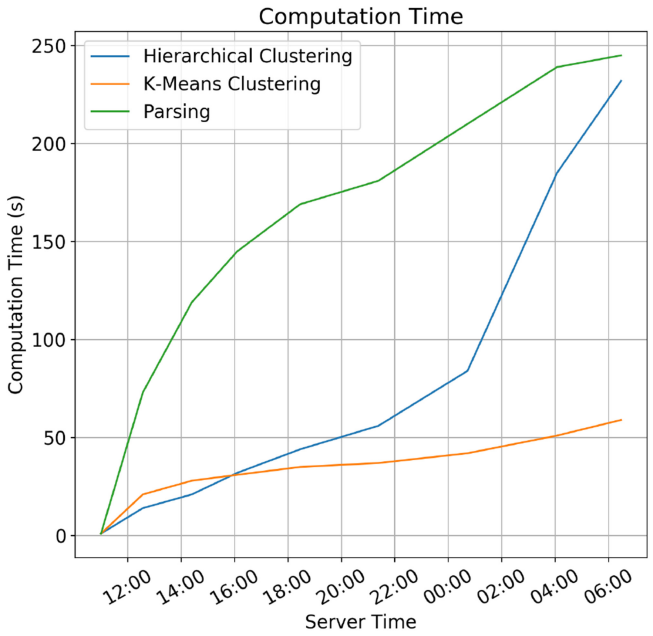


Fig. 7. Running times for query parsing, hierarchical, and k-means clustering.

for clustering processes in general, which need to compute and evaluate a full $O(N^2)$ pairwise distance matrix. Fortunately, the number of skeletons grows sub-linearly with the number of queries, and the overall scaling by time is roughly linear. We expect that this growth curve would eventually become sub-linear for longer traces. Consequently, even if the training data size increases in terms of timeframe, the training data size in terms of number of queries being clustered will not grow with the same rate.

7 DISCUSSION

We focused on modeling the insider threat to the databases while analyzing the computational complexity of the attacks, and defense mechanisms.

Our work paves the way for creating approximation approaches for computationally complex attack and defense models analyzed in this article. Such approximations are not guaranteed to provide optimal results; however, a computationally feasible approximation method can make NP–Complete problems tractable. Furthermore, a formal understanding of the complexity of the attacks will help with developing appropriate defense mechanisms.

The defense mechanisms discussed in Section 4 can further be improved by exploiting the sensitivity level of data accessed by the users or by constructing a *User Activity Graph*. The user activity graph is built with the historical actions of the user. It can be used to build normal behavior of users and predict the next move of the user with a method such as Hidden Markov Model [39]. The utilization of the probability of a user's next action can help the defense mechanism determine how likely an action will occur. One should also consider that building normal behavior models is effective, but benign users can also change their behavior over time [29].

Construction of key risk graphs helps understanding adversarial behavior, and defenders can utilize risk analysis to detect attacks. Attackers take risks to access sensitive data while trying to minimize the overall risk they take. This raises the question of how to identify sensitivity level of data in a database. One of the most obvious ways to determine the sensitivity level of data in a database is getting every column in a table evaluated by domain experts. For example, in a bank database, USER table might include *FirstName* and *LastName* columns along with *SSN*. While there is little to no risk of *FirstName* and *LastName* columns being queried, it might be a problem if *SSN* column is queried without legitimate basis. Another research direction here is to automatically identify sensitive information in a database.

However, while evaluating an activity to consider if it is a threat, the importance of the resource should not be the sole indicator of an attack. The workforce in the organization is designed to perform a specific duty that may require access to very sensitive data. Thus, the aim should be analyzing if the user actually requires that data to perform their task.

8 CONCLUSION

In this article, we first described the notion of user intent, which we can utilize in insider threat detection research, and we discussed the complexity of identifying the pseudo-intent of a user. We defined two attack models: (1) harvester attacks and (2) targeted attacks, while discussing the complexity of both attack and defense mechanisms in the literature. We then discussed the key risk problem. Finally, we showed that the defense mechanisms can, in fact, be used in production systems in terms of running time.

We plan to extend our work in several directions: First, we will collect query datasets from various environments to empirically analyze the effect of handwritten and generated query sets. Second, we will seek real attack cases where we can investigate anomalies they cause not only in the intent, but also in the system they occur. Third, we will develop a data anonymization framework to protect the privacy and identity of the users during routine investigations. Last, we plan to develop a real-world deployable software to detect insider threats to increase industry collaborations and engagement.

ACKNOWLEDGMENTS

We would like to thank all the members of Dr. Oliver Kennedy's research group for their role in fixing the bugs and testing JSQLParser.

REFERENCES

[1] Rakesh Agrawal, Paul Bird, Tyrone Grandison, Jerry Kiernan, Scott Logan, and Walid Rjaibi. 2005. Extending relational database systems to automatically enforce privacy policies. In *Proceedings of the International Conference on Data Engineering (ICDE'05)*.

[2] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. 2014. Similarity measures for OLAP sessions. *Knowl. Inf. Syst.* 39 (2014), 463–489.

[3] Robert H. Anderson and Richard Brackney. 2005. Understanding the insider threat. RAND Corporation. Retrieved from http://www.rand.org/pubs/conf_proceedings/CF196.

[4] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. 2006. Clustering-based materialized view selection in data warehouses. In *Proceedings of the European Conference on Advances in Databases and Information Systems*.

[5] Mikhail A. Babin and Sergei O. Kuznetsov. 2010. Recognizing pseudo-intents is coNP-complete. In *Proceedings of the International Conference on Concept Lattices and their Applications*.

[6] Matt Bishop, Sophie Engle, Deborah A. Frincke, Carrie Gates, Frank L. Greitzer, Sean Peisert, and Sean Whalen. 2010. A risk management approach to the "insider threat."*Insider Threats in Cyber Security*. Springer, 115–137.

[7] Matt Bishop and Carrie Gates. 2008. Defining the insider threat. In *Proceedings of the Conference on Cyber and Information Security Research*.

[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3 (2009), 15.

[9] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th ACM Symposium on Theory of Computing*. ACM, 77–90.

[10] Gloria Chatzopoulou, Magdalini Eirinaki, Suju Koshy, Sarika Mittal, Neoklis Polyzotis, and Jothi Swarubini Vindhiya Varman. 2011. The QueRIE system for personalized query recommendations. *IEEE Data Eng. Bull.* 34, 2 (2011).

[11] Ramkumar Chinchani, Anusha Iyer, Hung Q. Ngo, and Shambhu Upadhyaya. 2005. Towards a theory of insider threat assessment. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'05)*.

[12] Cisco Systems, Inc. 2017. 2017 annual cybersecurity report. (Jan. 2017). https://newsroom.cisco.com/press-release-content?articleId=1818259 and accessedDate=06/15/2019.

[13] E. Costante, S. Vavilis, S. Etalle, J. den Hartog, M. Petković, N. Zannone, et al. 2013. Database anomalous activities detection and quantification. In *Proceedings of the International Conference on Information Security and Cryptography (SECRYPT'13)*.

[14] Felix Distel and Barış Sertkaya. 2011. On the complexity of enumerating pseudo-intents. *Discr. Appl. Math.* 159, 6 (2011), 450–466.

[15] Uriel Feige. 1998. A threshold of Ln N for approximating set cover. *J. ACM* 45, 4 (July 1998), 634–652. DOI: https://doi.org/10.1145/285055.285059

[16] David Ferraiolo and John Barkley. 1997. Specifying and managing role-based access control within a corporate intranet. In *Proceedings of the ACM Role-based Access Control Workshop*.

[17] Wolfgang Gatterbauer. 2011. Databases will visualize queries too. *pVLDB* 4, 12 (2011) 1498–1501.

[18] Duc Ha, Shambhu Upadhyaya, Hung Ngo, Suranjan Pramanik, Ramkumar Chinchani, and Sunu Mathew. 2007. Insider threat analysis using information-centric modeling. In *Proceedings of the IFIP International Conference on Digital Forensics*. Springer, 55–73.

[19] Steven Homer and Alan L. Selman. 2011. *Computability and Complexity Theory*. Springer Science & Business Media.

[20] Ashish Kamra, Evimaria Terzi, and Elisa Bertino. 2007. Detecting anomalous access patterns in relational databases. *VLDBJ* 17 (2007), 1063–1077.

[21] Nikolaos Karapanos, Alexandros Filios, Raluca Ada Popa, and Srdjan Capkun. 2016. Verena: End-to-end integrity protection for web applications. In *Proceedings of the Symposium on IEEE Security and Privacy*.

[22] Muhammad Imran Khan, Barry O'Sullivan, and Simon N. Foley. 2017. A semantic approach to frequency based anomaly detection of insider access in database management systems. In *Proceedings of the 12th International Conference on Risks and Security of Internet and Systems*. 18–28. DOI: https://doi.org/10.1007/978-3-319-76687-4_2

[23] Muhammad Imran Khan, Barry O'Sullivan, and Simon N. Foley. 2018. Towards modelling insiders behaviour as rare behaviour to detect malicious RDBMS access. In *Proceedings of the IEEE International Conference on Big Data (Big Data'18)*. IEEE, 3094–3099.

[24] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. 2010. SnipSuggest: Context-aware autocompletion for SQL. *Proceedings of the VLDB Endowment* 4, 1 (2010), 22–33.

[25] Phokion Kolaitis. 2010. Relational databases, logic, and complexity. In *Proceedings of the Sino-European Winter School in Logic, Language and Computation*. Retrieved from https://users.soe.ucsc.edu/ kolaitis/talks/gii09-final.pdf.

[26] Gokhan Kul, Duc Luong, Ting Xie, Patrick Coonan, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. 2016. Ettu: Analyzing query intents in corporate databases. In *Proceedings of the International Conference on World Wide Web*.

[27] Gokhan Kul, D. T. A. Luong, Ting Xie, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. 2018. Similarity metrics for SQL query clustering. *IEEE Trans. Knowl. Data Eng.* 30, 12 (Dec. 2018), 2408–2420. DOI: https://doi.org/10.1109/TKDE.2018.2831214

[28] Gokhan Kul and Shambhu Upadhyaya. 2015. Towards a cyber ontology for insider threats in the financial sector. *J. Wirel. Mob. Netw. Ubiq. Comput. Depend. Applic.* 6, 4 (2015), 64–85.

[29] G. Kul, S. Upadhyaya, and V. Chandola. 2018. Detecting data leakage from databases on Android apps with concept drift. In *Proceedings of the 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE'18).* 905–913. DOI: https://doi.org/10.1109/TrustCom/BigDataSE.2018.00129

[30] Gokhan Kul, Shambhu Upadhyaya, and Andrew Hughes. 2017. Complexity of insider attacks to databases. In *Proceedings of the International Workshop on Managing Insider Security Threats (MIST'17).* ACM, New York, NY, 25–32. DOI: https://doi.org/10.1145/3139923.3139927

[31] Deloitte Development LLC. 2019. The Deloitte 2019 future of cyber survey. Deloitte. Retrieved from https://www2.deloitte.com/us/en/pages/advisory/articles/future-of-cyber-survey.html.

[32] Vitor Hirota Makiyama, M. Jordan Raddick, and Rafael D. C. Santos. 2015. Text mining applied to SQL queries: A case study for the SDSS SkyServer. In *Proceedings of the International Conference on Information Management and Big Data (SIMBig'15).*

[33] Sunu Mathew, Michalis Petropoulos, Hung Q. Ngo, and Shambhu Upadhyaya. 2010. A data-centric approach to insider attack detection in database systems. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses.*

[34] Andrew Stephen McGough, Budi Arief, Carl Gamble, David Wall, John Brennan, John Fitzgerald, Aad van Moorsel, Sujeewa Alwis, Georgios Theodoropoulos, and Ed Ruck-Keene. 2015. Ben-ware: Identifying anomalous human behaviour in heterogeneous systems using beneficial intelligent software. *J. Wirel. Mob. Netw. Ubiq. Comput. Depend. Applic.* 6, 4 (2015), 3–46.

[35] Muhammad Naveed, Seny Kamara, and Charles V. Wright. 2015. Inference attacks on property-preserving encrypted databases. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'15).*

[36] N. Nguyen, P. Reiher, and G. H. Kuenning. 2003. Detecting insider threats by monitoring system call activity. In *Proceedings of the IEEE SMC Information Assurance Workshop (IAW'03).*

[37] Ponemon Institute. 2017. 2017 Cost of Cyber Crime Study: Global. (Oct. 2017). https://www.accenture.com/t20170926T072837Z__w__/us-en/_acnmedia/PDF-61/Accenture-2017-CostCyberCrimeStudy.pdf and accessed Date=06/16/2019.

[38] PwC. 2017. The global state of information security survey 2018. (Oct. 2017). https://www.pwc.com/us/en/services/consulting/cybersecurity/library/information-security-survey.html and accessedDate=04/12/2019.

[39] Tabish Rashid, Ioannis Agrafiotis, and Jason R. C. Nurse. 2016. A new take on detecting insider threats: Exploring the use of hidden Markov models. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats.* 47–56.

[40] Malek Ben Salem, Shlomo Hershkop, and Salvatore J. Stolfo. 2008. A survey of insider attack detection research. In *Insider Attack and Cyber Security.* Springer, 69–90.

[41] A. Sallam and E. Bertino. 2017. Detection of temporal insider threats to relational databases. In *Proceedings of the IEEE International Conference on Collaboration and Internet Computing (CIC'17).* 406–415. DOI: https://doi.org/10.1109/CIC.2017.00058

[42] Asmaa Sallam and Elisa Bertino. 2018. Detection of temporal data ex-filtration threats to relational databases. In *Proceedings of the IEEE International Conference on Collaboration and Internet Computing (CIC'18).* IEEE, 146–155.

[43] Ameya M. Sanzgiri and Shambhu Upadhyaya. 2011. Feasibility of attacks: What is possible in the real world—A framework for threat modeling. In *Proceedings of the International Conference on Security and Management (SAM'11).*

[44] William Stallings. 2017. *Cryptography and Network Security: Principles and Practices, 7th Edition.* Pearson.

[45] Yuqing Sun, Haoran Xu, Elisa Bertino, and Chao Sun. 2016. A data-driven evaluation for insider threats. *Data Sci. Eng.* 1, 2 (2016), 73–85. DOI: https://doi.org/10.1007/s41019-016-0009-x

[46] Yuqing Sun, Haoran Xu, Elisa Bertino, and Chao Sun. 2016. A data-driven evaluation for insider threats. *Data Sci. Eng.* 1, 2 (2016), 73–85. DOI: https://doi.org/10.1007/s41019-016-0009-x

[47] Shiyuan Wang, Divyakant Agrawal, and Amr El Abbadi. 2010. HengHa: Data harvesting detection on hidden databases. In *Proceedings of the ACM Conference on Cloud Computing Security.*

[48] X. Yang, C. M. Procopiuc, and D. Srivastava. 2009. Recommending join queries via query log analysis. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'09).*

[49] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2015. IntegriDB: Verifiable SQL for outsourced databases. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security.* ACM, 1480–1491.

Received September 2019; revised March 2020; accepted March 2020