Complexity of Insider Attacks to Databases

Gokhan Kul, Shambhu Upadhyaya, Andrew Hughes
Department of Computer Science and Engineering
University at Buffalo, SUNY
338 Davis Hall
Buffalo, New York 14260-2500
{gokhanku,shambhu,ahughes6}@buffalo.edu

ABSTRACT

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 paper, we first define the notion of user intent, and construct a model for the most common threat scenario used in the literature 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 is coNP-Complete in this domain, 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 about the general defense mechanisms against the modeled threats, and show that countering against the harvester insider attack model takes quadratic time while countering against the targeted insider attack model can take linear to quadratic time depending on the strategy chosen. Finally, we analyze the adversarial behavior, and show that launching an attack with minimum risk is also an NP-Complete problem.

CCS CONCEPTS

• **Security and privacy** → **Database activity monitoring;** • **Theory of computation** → *Algebraic complexity theory;* • **Information systems** → Structured Query Language;

KEYWORDS

Complexity Analysis; Insider Threat; Query Intent; Query Logs; Threat Modeling

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 still persists [12]. One of the most critical types of cyber attack is called *insider attack*, which occurs when employees misuse their legitimate access rights, or gain unauthorized access to a resource, such as file systems and servers [6]. Unauthorized account openings, and money transfers, identity theft, and credit fraud are just a few other well-known examples that this attack type includes.

The "2016 Global State of Information Security Survey" [30] states that in 2015, 34% of the security incidents originated from current employees while 29% of the incidents originated from former employees. Another study in 2016 [29] shows that the percentage of organizations that experienced at least one insider incident increased from 35% to 41%. A 2015 study [28] 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. Another study in 2014 [7] found out that although 37% of organizations in the U.S. have experienced such an incident, and only 3% of them were reported to the authorities due to lack of evidence.

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. Lastly, a new legitimate activity from a legitimate actor can be perceived as an anomaly although the intent is benign.

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 [31]. These resources can be shell commands [27], file accesses [25], and SQL queries [18, 24]. Another branch focuses on psychological factors instead of resource usage [5].

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 clearly lacks 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

MIST'17, October 30, 2017, Dallas, TX, USA.
 ISBN 978-1-4503-5177-5/17/10...$15.00
DOI: http://dx.doi.org/10.1145/3139923.3139927

evaluate the time complexity of launching an attack using the modeled threats, and discuss the defense mechanisms to counter these while considering the time complexity.

Concretely, the contributions of this paper are:

(1) Defining the notion of intent model for database users,
(2) Building a threat model for insider attacks in the context of relational databases,
(3) Analyzing counter actions against the modeled threat, and
(4) Providing a computational complexity analysis for both the threat and defense mechanisms

based on the state of the art in this area.

This paper is organized as follows. We start by mathematically formulating the intent model in Section 2. We then describe the threat model in Section 3. We discuss other factors that can be utilized in both threat and defense models in Section 4. Finally, we conclude, and briefly present our future work in Section 5.

2 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 [22]. 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 [16]. 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 [3], workload analysis [2, 23], query recommendation [10, 20, 36], and security purposes [18]. As the need to access more complex information, the construction of the query intrinsically gets 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].

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. Of course, the results of queries also depend on the data stored in the database. For example, the queries

```
SELECT * FROM user WHERE username LIKE "A%"
SELECT * FROM user
```

will produce exactly the same result if all the `username` values in the user table start with "A." 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 $\mathtt{g}$ with the attributes satisfied by $\mathtt{g}$. In order to express that an object $\mathtt{g}$ is in relation $\mathbb{I}$ with an attribute $\mathtt{m}$, we write $\mathtt{g}\mathbb{I}\mathtt{m}$ [4].

Identifying if a subset of attributes is a pseudo-intent is shown to be coNP-Complete [4]. 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.

Definition 2.1 (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 $\phi = \{r_1, r_2, ..., r_{|\phi|}\}$.

This definition directly conforms with the feature extraction methods described above where r_i is the extracted query features.

Definition 2.2 (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,

$$A_u^T = a_u^{t_0}\phi, a_u^{t_1}\phi, ..., a_u^{t_n}\phi \tag{1}$$

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 [22]. These workloads can be utilized to create a chain of tasks for each user.

Definition 2.3 (User Activity Graph). User activity graph G is a directed and weighted graph that shows the historical navigation of the user between resources. $G = V, E$ where V is the set of all possible activities performed by the user and E is the set of navigations from one activity to the other.

There can be two types of weights: (1) the number of different items between two sets, and (2) cumulative probability of the next activity where the probability is calculated by the division of the number of occurrences of an item to the total item occurrences.

3 THREAT MODEL

Stallings [33] classifies security 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 paper, we survey the research focusing on detecting data leakage by insiders using query logs. The state of the art of this research targets *passive attackers*: this type of attackers query the database to extract sensitive information. The threat model assumes that the insider does not tamper with the data, or execution on the client-side application. 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.

We do not address *active attackers* who tamper with the data, or query results because we aim to address the threat posed by insiders who try to steal information, in order to prevent identity theft, and information leakage. We also do not address snapshot attacks where the attacker copies the database server, or the database instance completely without getting caught by any logging system. Both attack and defense strategies for this attack type are completely different. If such an attack is successful, the adversary can access the copy of the database instance offline without having to go through the query monitor. To address data tampering by an insider, the construction can be supported with integrity verification techniques [19], and authenticated data structures [37].

We consider two passive attack models; harvester (*a.k.a. aggregate*) and targeted (*a.k.a. individual*) attackers [26]. 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, on the other hand, an adversary who accesses certain information without needing to know for legitimate purposes, but chooses the attack parameters carefully to avoid being detected.

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.

If an adversary can issue a query and not get detected by the defense mechanism, the query evaluation on just one table takes linear time. 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 $O|R|$ 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 $Olog|R|$ 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 [21].

3.1 Architecture

We utilize the query intent modeling while monitoring database activity to flag potential insider attacks. This construction aims to identify attackers to the database system. The overview of the architecture can be seen in Figure 1. 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 just observes the queries that are issued to the database, and it doesn't block or change the queries. Any query that is issued to the database is captured by the query monitor, 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, the query monitor just observes the interaction between users and the database, but never modifies the query results or database records.

Next, we define the threat model this construction is built against.

3.2 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.* [35]. 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.

3.2.1 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 don't 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 for each table, and can retrieve all the database by issuing number of tables $|R|$ queries.

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

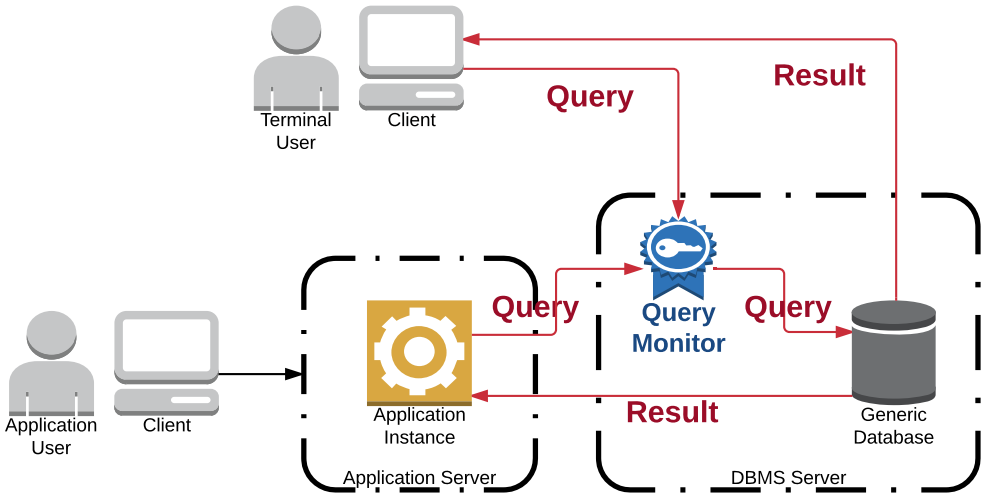

Figure 1: An architecture of query monitoring

only 3 queries: `SELECT * FROM customer`, `SELECT * FROM account`, and `SELECT * FROM customer_accounts`.

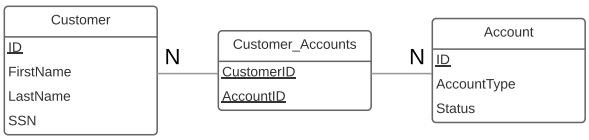

Figure 2: `MY_BANK` database diagram

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 $O|R|$.

3.2.2 The harvester defense model. Analyzing queries mostly relies on the structure of queries [18] 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 2, and, hence, the features in the intent set can be used to measure diversity. Data-centric query comparison [24], on the other hand, 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 broadness.

For instance, Makiyama *et al.* [23] 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 [18]. 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 $O1$. 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 On, where n is the number of queries issued. To compute the diversity, we compare the resources in ϕ_{ALL} with all the resources possible (S) in the database. The time complexity of this operation is $O|\phi_{ALL}| \cdot |S|$.

3.3 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 [18, 22, 34].

3.3.1 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 insider should be precise about the information that they need. 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 [23], the access policy can prevent certain information from being used together [1], or accessing too much information can raise an alert [15]. 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 presented in Section 2 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, 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 [17] and reduces in polynomial time to Intent Set, Intent Set is NP–Complete. □

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. That is to say, a general intuition this proof conveys is that, without having knowledge about the contents of the database, and having k number of constraints due to the risk of getting caught if too many resources are used, a query should be constructed very carefully by an attacker. As the user intent grows, restricting k will make it more challenging to construct a query that provides access to the target data.

Example. Non-admin users

For illustration purposes, let's 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's assume that the current credentials that Ashley uses limit her to access at most 4 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

$$I = \{username, rank, <>, "admin"\}$$

Consequently, she would be using 4 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 4 resources, she won't be able to issue this query without getting caught.

3.3.2 The targeted defense model. The distinctive traits of targeted attacks are that the adversaries access information without needing to know, and deviate from their normal behavior. There are two approaches to prevent 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

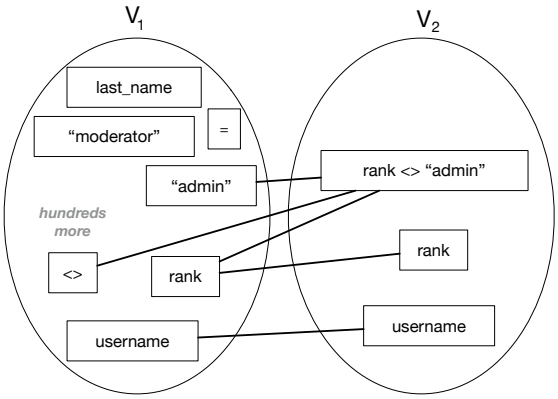

Figure 3: Intent Set for `SELECT username, rank FROM user WHERE rank <> "admin"`

the forbidden behavior defined in the rule set. This check is usually straight-forward, and takes linear time.

Anomaly detection mechanism, on the other hand, forms *user profiles* in order 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 [18, 34].

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].

3.4 Analysis of Adversarial Behavior

An adversary launches 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 [32]. 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 in order to reduce the risk of getting caught. However, while trying to reduce the risk, the adversary still needs 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 an adversary 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

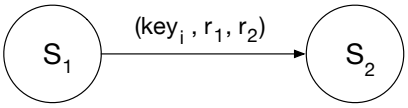

Figure 4: Key risk representation

Definition 3.1 (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 one state to another.

$KG = V, E; K, V_0, V_S, \pi, \delta$ where V represents the set of all possible states, and E represents the set of navigations from one state to the another. K 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, $\pi : V \rightarrow K$ is a function that assigns keys to the states, and lastly, $\delta : E \rightarrow K \times \mathbb{N} \times \mathbb{N}$ 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.

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 credit collaborating with her contact, Dan, in the credit department. Then pay off the credit using the money in the account. 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 in 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 paper works, she evaluates both strategies. The first strategy involves looking up an address which 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), on the other hand, is just a very common procedure while paying off a loan, hence, she finds it less risky (r_1).

4 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 paper. Such approximations are not guaranteed to provide optimal results; however, a computationally feasible approximation method can make NP–Complete problems tractable. A better understanding of the complexity of the attacks will help with developing appropriate defense mechanisms.

The defense mechanisms discussed in Section 3 can further be improved by exploiting the sensitivity level of data accessed by the users, or by constructing a *User Activity Graph*.

One of the most obvious ways to determine the sensitivity level of data in a database is that 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.

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.

The user activity graph is built with the historical actions of the user, and could be used to predict the next move of the user with a method such as Hidden Markov Model. The utilization of the probability of the next action of a user can help the defense mechanism determine how likely an action will occur.

5 CONCLUSION

In this paper, 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 then defined two attack models: (1) harvester attacks, and (2) targeted attacks, while discussing the complexity of both attack, and defense mechanisms in the literature. Finally, we discussed the key risk problem.

We plan to extend our work in several directions: First, we will seek to develop approximation algorithms for both intent detection, and targeted attack model. Second, we will collect query datasets to empirically show the efficacy of both attack, and defense models defined in this work. Lastly, we will analyze the effect of approximation algorithms, and evaluate their real-world plausibility with the corresponding defense strategies.

ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under award number CNS - 1409551. Usual disclaimers apply.

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 *ICDE*.
[2] Julien Aligon, Matteo Golfarelli, Patrick Marcel, Stefano Rizzi, and Elisa Turricchia. 2014. Similarity measures for OLAP sessions. *Knowledge and information systems* (2014).
[3] Kamel Aouiche, Pierre-Emmanuel Jouve, and Jérôme Darmont. 2006. Clustering-based Materialized View Selection in Data Warehouses. In *ADBIS*.
[4] Mikhail A Babin and Sergei O Kuznetsov. 2010. Recognizing Pseudo-intents is coNP-complete. In *CLA*.
[5] 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* (2010), 115–137.
[6] Matt Bishop and Carrie Gates. 2008. Defining the Insider Threat. In *CSIIRW*.
[7] CERT Insider Threat Center. 2014. 2014 U.S. State of Cybercrime Survey. (July 2014).
[8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 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 ninth annual 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.* (2011).
[11] Ramkumar Chinchani, Anusha Iyer, Hung Q Ngo, and Shambhu Upadhyaya. 2005. Towards a theory of insider threat assessment. In *DSN*.
[12] Cisco Systems, Inc. 2017. 2017 Annual Cybersecurity Report. (January 2017).
[13] E. Costante, S. Vavilis, S. Etalle, J. den Hartog, M. PetkoviÄĞ, and N. Zannone. 2013. Database anomalous activities detection and quantification. In *SECRYPT*.
[14] Felix Distel and Barış Sertkaya. 2011. On the complexity of enumerating pseudo-intents. *Discrete Applied Mathematics* 159, 6 (2011), 450–466.
[15] David Ferraiolo and John Barkley. 1997. Specifying and managing role-based access control within a corporate intranet. In *ACM RBAC*.

[16] Wolfgang Gatterbauer. 2011. Databases will visualize queries too. *pVLDB* (2011).
[17] Steven Homer and Alan L Selman. 2011. *Computability and complexity theory*. Springer Science & Business Media.
[18] Ashish Kamra, Evimaria Terzi, and Elisa Bertino. 2007. Detecting anomalous access patterns in relational databases. *VLDBJ* (2007).
[19] Nikolaos Karapanos, Alexandros Filios, Raluca Ada Popa, and Srdjan Capkun. 2016. Verena: End-to-end integrity protection for web applications. In *IEEE Security and Privacy*.
[20] Nodira Khoussainova, YongChul Kwon, Magdalena Balazinska, and Dan Suciu. 2010. SnipSuggest: context-aware autocompletion for SQL. *pVLDB* (2010).
[21] Phokion Kolaitis. 2010. Relational Databases, Logic, and Complexity. *Sino-European Winter School in Logic, Language and Computation, Guangzhou, China* (2010). https://users.soe.ucsc.edu/~kolaitis/talks/gii09-final.pdf
[22] Gokhan Kul, Duc Luong, Ting Xie, Patrick Coonan, Varun Chandola, Oliver Kennedy, and Shambhu Upadhyaya. 2016. Ettu: Analyzing Query Intents in Corporate Databases. In *WWW Companion*.
[23] Vitor Hirota Makiyama, M Jordan Raddick, and Rafael DC Santos. 2015. Text Mining Applied to SQL Queries: A Case Study for the SDSS SkyServer. In *SIMBig*.
[24] Sunu Mathew, Michalis Petropoulos, Hung Q. Ngo, and Shambhu Upadhyaya. 2010. A Data-centric Approach to Insider Attack Detection in Database Systems. In *RAID*.
[25] 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. Benware: Identifying Anomalous Human Behaviour in Heterogeneous Systems Using Beneficial Intelligent Software. *JOWUA* (2015).
[26] Muhammad Naveed, Seny Kamara, and Charles V Wright. 2015. Inference attacks on property-preserving encrypted databases. In *ACM CCS*.
[27] N. Nguyen, P. Reiher, and G. H. Kuenning. 2003. Detecting insider threats by monitoring system call activity. In *IEEE IAW*.
[28] Ponemon Institute. 2015. 2015 Cost of Cyber Crime Study: Global. (October 2015).
[29] Ponemon Institute. 2016. 2016 Cost of Cyber Crime Study & the Risk of Business Innovation. (October 2016).
[30] PwC. 2015. The Global State of Information Security Survey 2016. (October 2015).
[31] 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.
[32] Ameya M Sanzgiri and Shambhu Upadhyaya. 2011. Feasibility of attacks: What is possible in the real world–a framework for threat modeling. In *SAM*.
[33] William Stallings. 2017. *Cryptography and network security: principles and practices, 7th Edition*. Pearson.
[34] Yuqing Sun, Haoran Xu, Elisa Bertino, and Chao Sun. 2016. A Data-Driven Evaluation for Insider Threats. *Data Science and Engineering* 1, 2 (2016), 73–85. DOI:http://dx.doi.org/10.1007/s41019-016-0009-x
[35] Shiyuan Wang, Divyakant Agrawal, and Amr El Abbadi. 2010. HengHa: Data Harvesting Detection on Hidden Databases. In *ACM CCSW*.
[36] X. Yang, C. M. Procopiuc, and D. Srivastava. 2009. Recommending Join Queries via Query Log Analysis. In *IEEE ICDE*.
[37] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2015. Integridb: Verifiable SQL for outsourced databases. In *ACM CCS*.