**How Are We Keeping Our Data Safe : A Look at Security Techniques For Distributed Databases**

Mardel Maduro - Athabasca University Comp 418 Research Project

**Abstract**

This paper presents an overview of the types of attacks that are currently affecting distributed databases around the globe. It covers how these attacks are classified based on how they are carried out, explains how each type of attack works and covers the techniques currently in use to handle the threats that arise due to the attacks. The information is then explored, pointing out the pro and cons on various techniques used for various attacks types to ascertain which types of attacks prove to be the most threatening and which techniques the most effective.

**1. Introduction**

With the growing focus of making use of data for the purposes of Data Science and Machine Learning, the need to keep this data secure also grows. Distributed databases have become the go to way for housing all of the data that organizations see as necessary to keep record of. These large systems tend to contain many different types of machines, from many different physical locations all communicating with each other over a network. With all these added complexities to a system, it is a necessity to ensure that we protect the vulnerabilities that arise.

The main goal of this research paper is to dive into the types of security issues we are currently facing with our implementations of distributed database systems as well as to have a look at the current techniques that are being used to combat these issues. The aim is to look at which issues are the most threatening and which techniques work best based on our current knowledge and capabilities.

Before describing the techniques to secure databases, it is preferable to describe the attacks which can be performed on the databases. The major attacks on databases can be categorized as shown in Fig. 1, which is inference, active and passive attacks in databases and SQLIA. [1]

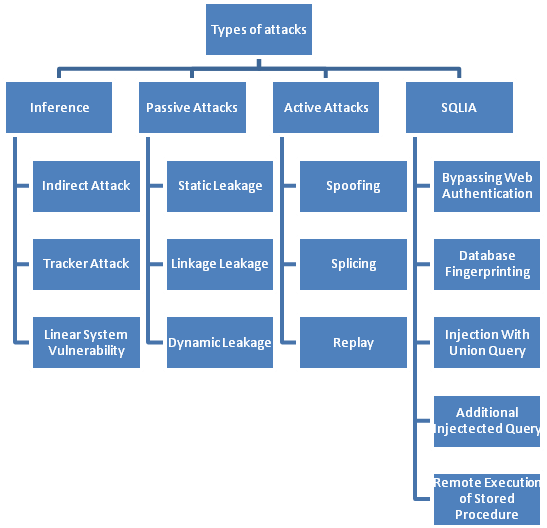


Fig. 1 Types of Attacks on Databases [1]

By using theses categories as a starting point, we will explore each type of vulnerability as well as the mechanisms or techniques we must put in place to protect our DDBs against them.

**2. Methods**

**2.1 Inference Attacks**

Inference Attacks occur when a malicious user infers some protected or private data without directly accessing it [2]. This is done by using the data that is available to them to make informed guesses as to what said inaccessible data might contain. This is obviously an issue since if you have many users who may have access to the information in your DDB, the chances of someone with legitimate access using it for illegitimate reasons can easily arise. These types of attacks can usually be conceptualized using the AERIE model (Activities-Entity-Relationships-Inference-Effects). This model, created by a research project done at the University of Alabama, is used to describe the types of targeting used when taking available information to infer unavailable information. According to this model there are 6 types of inference attacks that commonly take place.

1) Entity Materialization : this type of inference attack represents the detection of the existence on an entity [2,3]

2) Activity Materialization : this type of inference attack represents the detection of the existence of an activity [2,3]

3) Entity-Entity Relationship : this type of inference attack represents the detection of a relationship between two entities. [2,3]

4) Activity-Activity Relationship : this type of inference attack represents the detection of a relationship between two activities. [2,3]

5) Entity-Activity Relationship : this type of inference attack represents the detection of a relationship between and entity and an activity.[2,3]

6) Relationship-Relationship Relationship : this type of inference attack represents the detection of a relationship between two relationships.[2,3]

In order to tackle these kind of attacks, we need to be able to understand the logic behind how the adversary may come to make these inferences. One way to do this is to make use of the inference rules. These rules are not only used to identify if there is a possibility for an attacker to infer unavailable information, but is also used as a basis for the prevention system that can be implemented to prevent said attacks from occurring. These rules, created by Yip and Levitt in the 1980s, identified the use of intersection, union and difference relationships between multiple queries by adversaries to infer protected information[2]. The rules are as follows:

1) Slit Queries – These types of queries occur when a query can be split into 2 smaller inferred queries[2]. This can be thought of as cross checking 2 queries to infer data based on what records they intersect on.

2) Subsume Queries – These occur when 2 queries are related in a way such that all results of the first query have some common attribute, which can then be used to infer data from the second query [2]. If person X has attribute A, and everyone with attribute A, has attribute B, we can infer that person X has attribute B.

3) Unique Characteristics Inference – These work by exploiting the fact that a record in the system may have a unique value for an attribute. Based on that attribute the attacker can make inferences by comparing a query that would include everyone and another that would exclude that unique value.

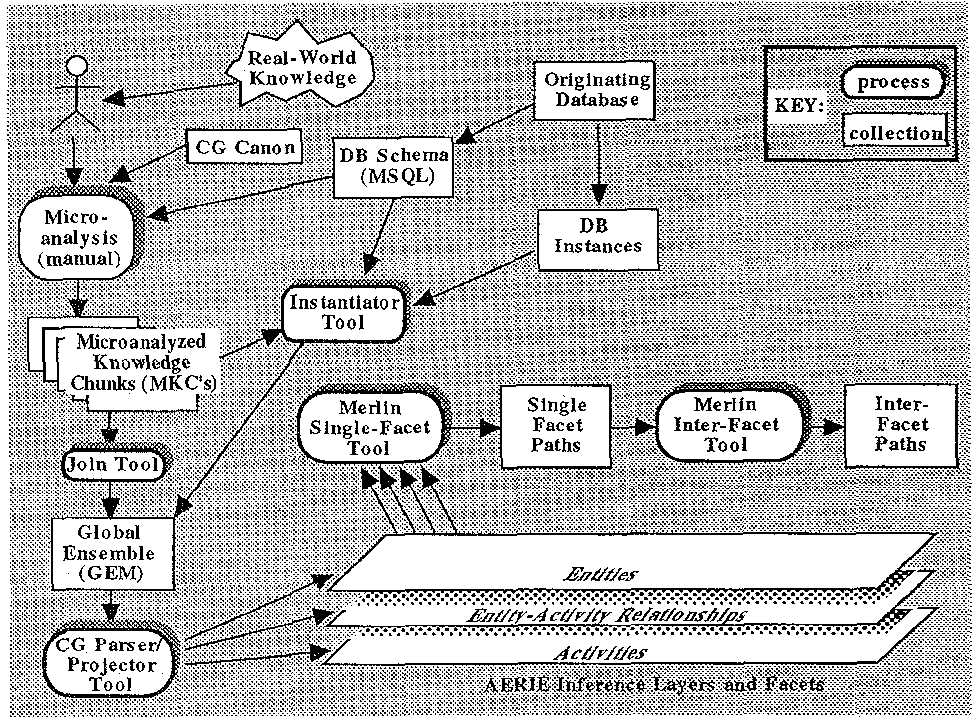
4) Overlapping Inference – Inferring data by the using of the intersection of multiple queries to find a specific record.

5) Complementary Inference – The opposite to overlapping, instead uses the difference of multiple queries to find a specific record. [2]

6) Functional Dependency Inference – The using of a primary key within a database to access other data. Ex. Using a student ID# to gain access to a students name, address etc.

Though incomplete and insufficient, these rules are sound and allow for a starting point for detecting basic weaknesses to inference attacks[2].

Another solution for dealing with inference attacks comes in the form of Wizard [4]. Wizard is a database inference analysis and detection system. Created by the people of the AERIE team at the University of Alabama, Wizard uses a tool they created called Merlin[4]. Merlin is used to analyze relations for the presence of second paths which can be used to detect inferences and detects how likely said second paths are to provide enough information to infer information accurately[4]. An image showing the overview of of the Wizard system can be seen below.



In addition to these systems based on automated analysis, a manual approach called Penetration and Fixing can also be used. As mentioned in [3], in this approach inference hypotheses are generated and classified based on severity. They are then validated, and an attempt to generalize them and make them into more generic problems is performed. Finally we take what we learned via steps 2 and 3 and either eliminate entirely or greatly reduce the chances of inference by classifying as much of the data as sensibly possible.

**2.2 Passive and Active Attacks**

Next on our list of security issues to take a look at are passive and active attacks. Passive attacks can be thought of as attacks to the database whereby the attacker only observes the data in the database[1,5]. These types of attacks can be carried out in one of the three of the following ways.

1) Static Leakage – In static leakage, information about the database plaintext values is obtained via the observation of database snapshots [1,5].

2) Linkage Leakage – In linkage leakage, information about the database plaintext values is obtained via the linking of database values with the positions of the values in the index.[1.5]

3) Dynamic Leakage – In dynamic leakage, information about the plaintext values in the database is obtained via the observation of changes made to the database over time. [1,5]

As for active attacks, we can think of them as attacks involving the modification of data within the database. These too come in 3 varieties.

1) Spoofing – Spoofing involves the taking of a cipher text value and replacing it with a generated one [5]

2) Splicing – Splicing involves the taking of a cipher text value and replacing it with another cipher text value.[5]

3) Replay – Replay involves that taking of a cipher text value and replacing it with an old version of the value that was either removed or modified [5].

These types of attacks can be carried out by authorized users in the system who may have more access to data than what is required for their job as well as external attackers who may gain access to the system via nefarious means. As such the first line of defence against these types of attacks is the formation of access control [6]. Access control is all about keeping data out of the sight of who doesn’t need to see it. This is done using a few different mechanisms.

1) Attribute Based Access Control – access is granted to a user via the use a policy which evaluates whether or not said user should have access based on a particular attribute.

2) Discretionary Access Control – access is granted to a user based on the creator of the data (relation)

3) History-Based Access Control – access is granted based on the real time evaluation of the user’s previous actions.

4) Identity Based Access Control – access is granted based on the needs of the individual user

5) Mandatory Access Control – access is granted based on the clearance level of the user.

6) Role Based Access Control – access is granted based on the users’s role in the organization

7) Rule-Based Access Control – access is granted to user based on a set of specific rules that have been put in place by the administrator.

By having these types of mechanisms in place we can reduce the chances of legitimate users executing active or passive attacks, whether accidentally or purposefully. This however isn’t always enough, as mentioned in [6], theses methods are expensive when it comes to maintaining access levels in a dynamic domain and also run the risk of reducing availability. Another technique that has been common in both academia and the corporate world is Query Monitoring. These techniques tend to fall into two categories;

1) Signature Based – this category covers query monitoring systems which work by setting forth a blacklist of dangerous or denied access patterns [6]. If a user does a specific sequence of queries we know to be a path to data that they shouldn’t be accessing, we have fair reason to believe that they are doing something heinous and we can lock them out of the system. A major disadvantage to this however is that since a list is needed to be made for comparison to check whether a query sequence is dangerous or not, only well know attacks can be prevented, as for an advantage once an alert is raised, it is very unlikely to be false.

2) Behaviour Based – in this category however, the monitoring systems work by looking at what is the normal types of queries to be made, anything that is out of the ordinary is perceived as a threat [6]. This can be seen as a plus or a minus. A plus since it means unknown attacks can be handle, but also a minus since it means that if a query is not dangerous but just uncommon it can raise red flags and can lock users out who are just trying to do their jobs.

**2.3 SQL Injection Attacks**

Lastly on our list of threats to take a look at are the SQL injection attacks. These types of attacks are considered the most dangerous of attacks on databases. They work by having attackers insert malicious SQL queries and can wreak havoc on database systems[1]. These come in five classifications.

1)Bypassing Web Authentication – In these types of SQL injection attacks, the adversary uses an input field to enter a syntactically true query which allows the attacker to access records for a user without having that user’s password. [1,7]

2) Database Fingerprinting – In these types of attacks, the adversary creates logically incorrect or illegal queries, which cause errors to occur in the database. These errors throw messages that can then be used by the attacker to guess what types of technologies the database is built on [1,7]. This can then be used to formulate other attacks, since the adversary can now create a plan of attack specifically for that system.

3) Injection with the union query – These work by the attacker gaining access to one table by injecting a query to it in another table. The developer intended for them to access table A, but the attack put in a query to table B within the ‘where’ part of the query to table A. This then causes the query to table A to display data from table B.

4) Damaging with additional injected query – In these types of attacks, the adversary causes damage to the data within the database by making use of the injected query. This is usually done by using drop or delete queries along with the intended query[1,7]. They cause major issues by allowing attackers to either remove records or entire tables from a database.

5) Remote execution of store procedures – In these kind of attacks the attacker causes harm by executing some unauthorized procedure stored within the system. They do this by injecting the call to the procedure within the SQL query that they entered into the input field. Once entered the procedure is then ran on the system storing the table that was queried. [7].

These SQL injection attacks can be handled in one of two way, detection or prevention.

SQL Injection detection technique are broadly split into 2 categories, Pre-generated Approaches and Post-generated Approaches [1].

Pre-Generated Approaches are all about finding vulnerabilities within the system during its development. According to [8], it all depends on the validity checking imposed by the developer and the effectiveness of the tools which they decide to use for detecting adversarial queries. Two of these approaches are Pixy and PQL.

Pixy is a static analysis tool which is used to detect the vulnerabilities of web apps by using flow sensitive, inter-procedural and context-sensitive data flow analysis to form statistical data for the points in the program[1,9]. Parse trees are then created, which are scanned using a taint-analysis tool to pinpoint which of the points in the program may be vulnerable to attacks [1]. However, this tool is open source, and as such is susceptible to attackers learning the weaknesses of the tool via exploration[1].

Another pre-generated approach is PQL (Program Query Language). In PQL, a predefined grammar is used to detect malicious queries by making use of a static and dynamic analyzer. The static one checks for matches using context-sensitive, flow-insensitive, inclusion based pointer alias analysis and the dynamic one uses a source program to try and catch violations as the program runs [10].

Post-Generated Approaches on the other hand analyze the SQL queries made before allowing them to interact with the database. These work using a few different techniques for query evaluation.[8]

The first of these techniques is the Positive Tainting and Syntax Aware Evaluation method. In this evaluation technique the system is provided with a set of valid input strings [1]. These strings are then used to determine whether a query being made by a user is a potential threat or not based on whether it contains strings not contained within the original list of valid ones. This list however is dependant on the knowledge of the developer [1], meaning gaps in his/her knowledge can result in vulnerabilities. Also these strings need to be stored somewhere[1], meaning that if not secured this list can be at risk and be modified by an attacker.

The second of these techniques is the Context Sensitive String Evaluation method. In this evaluation technique the system views all data introduced to the system by a user as non-trusted and application data as trusted [1]. The system then compares the two sets of data to determine which characters are valid for input and which aren’t [11]. Invalid characters are then removed from the alpha numeric identifiers [1]. Again the risk of vulnerabilities due to gaps in developer knowledge arise, since as mentioned in [1], the initialization of unsafe characters are dependant on the developer.

Lastly in the Post Generated approaches we have parse tree evaluation based on grammar. In this approach we create a parse tree using user input that is parsed using a predefined grammar [1]. The beginning and ending of the strings that are used for input are marked using special literals [1], and if the parsing of the resulting SQL query does not match the parse tree of the intended one, it is seen as an attempt to perform an SQL injection attack.

**3. Results and Findings**

As mentioned in the abstract and introduction, the purpose of this paper was to take a look at different classes of threats to Distributed Databases, how dangerous they can be in comparison to each other and what ways would be the best to solve them. Based on the research done, I have my opinions on the best ways to handle each class of attack and why I believe that to be the case.

Inference Attacks – Best solution would be a combination of Wizard and Penetration & Fixing. Using Wizard first to automate the process of finding as many vulnerabilities as possible, and then manually going through and seeing what vulnerabilities we can still find and fixing them. Though slow, it is faster than the fully manual approach of using Penetration & Fixing from the onset, but also more complete of a solution than a fully automated approach.

Passive and Active Attacks – The best solution for passive would be the implementation of Role-Based access control. In my opinion this approach provides a more effective way of making sure users only have access to what they need for their particular role in an organization. Though Identity-Based access control may be able to provide the same protection, it would also require making a profile for each person individually, which is very inefficient since many people with the same or similar roles would end up having the same profiles anyway. I also find it to be a better solution than query monitoring since the monitoring approaches require either very deep knowledge of attacks or a lot of time and patience to deal with false alarms. In addition to that, both approaches also require storage of a list for their implementation, a list which adds another point of vulnerability to the system.

SQL Injection Attack – The best solution here would be PQL. I consider PQL the best way to handle these types of attacks since it makes use of both a static analyzer prior to execution of the query and a dynamic analyzer which works as the query is being run. As such, it provides 2 mechanisms by which dangerous queries can be caught. It also doesn’t have the issues associated to weaknesses due to gaps in developer knowledge as two of the three post-generated approaches have.

Now as for what I have identified as the most serious of the attacks explored in this paper, I would have to agree with [1], SQL Injection Attacks are definitely the most dangerous of the bunch. As a group of threats that result in deletion of records or entire tables, the ability to access control procedure calls of the host system/s of a database and the ability to access user profiles without passwords they not only can result in ruined databases, denial-of-service , and unauthorized access to the system’s records but also can be used by attackers which are outside of the system to carry out the other forms of attacks mentioned in the paper.

**4. Related Works**

Distributed Database security has continued to be be a highly researched topic as time goes on and as systems continue to become larger and more complex. Various other papers similar to this one have covered the overview of security as well as provided deeper dives into specific types of attacks.

The one I found to be the closest to what I wanted to cover in this particular paper was the International Journal of Emerging Technology and Advanced Engineering’s ‘Review of Attacks on Databases and Database Security Techniques’[1]. As a paper that has been referenced multiple times within this one, it covered the basics types of attacks and how they are sub-divided. However, in my opinion, it didn’t cover inference, passive and active attacks in as much depth as this paper. On the other hand, it did explore the use of encryption for the protection of databases, which I chose not to cover since it is an overall security measure and not necessarily one created to deal with a specific attack type.

As for other papers that cover similar topics, a few provide deeper dives into specific types of attacks. These include :

For Inference type attacks, ‘Inference Attacks and Control on Database Structures’ and ‘Protecting Databases from Inference Attacks’, both of which provide deep dives into the specific types of inferences, how they are carried out and proposed techniques for dealing with them.

The same goes for ‘Database Security – SQL Injection’. This papers provided a comprehensive overview of SQL injection attacks, the tactics used to carry them out and the techniques used to detect and/or prevent them.

As papers that did such a good job of covering their particular areas of security they provided incredible insight that allowed for me to have a greater understanding of inference attacks and SQL injection attacks and be able to provide more meaningful breakdowns of the respective types of threats within this paper.

**5. Conclusion**

In conclusion, as databases continue to become more and more distributed and the push for open data for the use of machine learning and data science increases, we as database developers and administrators need to be mindful of the threats that loom over us and our systems. Various attacks were broken down, categorized and explored in this paper. As an overview of inference, passive, active and SQL injection attacks, the hope is that this paper can be used as a quick guide to security issues affecting distributed databases and how best to tackle them.

**References**

[1] Mr. Saurabh Kulkarni1 and Dr. Siddhaling Urolagin, “ Review of Attacks on Databases and Database Security Techniques,” ISSN 2250-2459, Volume 2, Issue 11, November 2012

[2] M. Turkanović, W. Družovec, M. Hölbl, "Inference Attacks and Control on Database Structures," TEM JOURNAL, 2015

[3] Hinke TH, Delugach HS, Wolf RP. Protecting databases from inference attacks. Computers and Security 1997;16(8):687–708

[4] Harry S. Delugach , Thomas H. Hinke, Wizard: A Database Inference Analysis and Detection System, IEEE Transactions on Knowledge and Data Engineering, v.8 n.1, p.56-66, February 1996 [doi>10.1109/69.485629]

[5] R. Kuar, Kiranpreet, P. Velma, “Survey on Database Security,” IJCA Journal, Volume 105-No. 10 , November 2014

[6] E. Costante , J. den Hartog , Milan Petković , S. Etalle , M. Pechenizkiy, A white-box anomaly-based framework for database leakage detection, Journal of Information Security and Applications, v.32 n.C, p.27-46, February 2017 [doi>10.1016/j.jisa.2016.10.001]

[7] Nikita Patel, Fahim Mohammed, Santosh Soni (2011) “SQL Injection Attacks: Techniques and Protection Mechanisms”, International Journal on Computer Science and Engineering (IJCSE) ISSN: 0975-3397 Vol. 3 No. 1 Jan 2011 199-203

[8] Debasish Das, Utpal Sharma, D. K. Bhattacharyya, “Rule based Detection of SQL Injection Attack”, International Journal of Computer Applications (0975 – 8887) Volume 43– No.19, April 2012.

[9] N. Jovanovic, C. Kruegel, and E. Kirda, “Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities,” in SP’06: Proceedings of the 27th IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2006.

[10] Michael Martin , Benjamin Livshits , Monica S. Lam, Finding application errors and security flaws using PQL: a program query language, Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, October 16-20, 2005, San Diego, CA, USA [doi>10.1145/1094811.1094840]

[11] Tadeusz Pietraszek , Chris Vanden Berghe, Defending against injection attacks through context-sensitive string evaluation, Proceedings of the 8th international conference on Recent Advances in Intrusion Detection, September 07-09, 2005, Seattle, WA [doi>10.1007/11663812\_7]